KU LEUVEN
FACULTEIT ECONOMIE EN
BEDRIJFSWETENSCHAPPEN

Advances in Process Mining:

Artificial Negative Events and Other Techniques

Proefschrift voorgedragen tot het behalen van de graad van Doctor in de Toegepaste Economische Wetenschappen

door

Seppe K.L.M. Vanden Broucke

Nummer 458 2014

Committee

Chairperson	Prof. dr. Martina Vandebroek	KU Leuven
Promotor	Prof. dr. Bart Baesens	KU Leuven
Co-Promotor	Prof. dr. Jan Vanthienen	KU Leuven
	Prof. dr. Josep Carmona	Universitat Politècnica de Catalunya
	Prof. dr. Arthur ter Hofstede	Queensland University of Technology
	Prof. dr. Jochen De Weerdt	KU Leuven
	Prof. dr. Guoqing Chen	Tsinghua University

To my parents, my sister, and Xinwei.
For their unwavering support and endless love.

Acknowledgments

"Don't walk behind me; I may not lead.
Don't walk in front of me; I may not follow.
Just walk beside me and be my friend."
– Albert Camus

A wise man—who most likely heard it from another wise person, as such things tend to go—once told me that a PhD journey feels not completely unlike a walk through the desert, in search of a beckoning light which calls you in the distance. To get there, however, one has to face many trials and tribulations, stumbling, falling in the sand, getting up and falling again. I'll stop here, but will add to this the fact that the journey is in large part about the encounters with fellow "nomads". Some of these fellow travelers you only meet for a few steps. Others join and walk with you for longer. Some of them are of quiet demeanor, whereas others have much to share. Some of them are there to pick you up and get you back on your feet just as you thought you'd fallen for the last time. In everything we undertake in life, it serves well to pause and be thankful for those around you. Although my name is the only one printed on the cover of this dissertation, it is by no means a solitary effort, and it could not have been completed without the support of others. I thus wish to dedicate a few pages to explicitly thank those kind fellow spirits who were with me "on the road", to use the famous words of Jack Kerouac.

First of all, it is both traditional and appropriate to thank my promotors, Prof. Bart Baesens and Prof. Jan Vanthienen. I believe the term "advisor" to be more fitting in this context, as both of them have been amazing sources of advice throughout the past three years. I immensely appreciate the freedom you both gave me in pursuing my research and I greatly value having gotten the opportunity to learn from and work with you both. Jan, thank you for the chats on research culture, the hell that is bureaucracy, and the good old days of IT research, back when it was okay to spend a week on learning that new crazy insert-buzzword-here-oriented language just out of personal interest. We need this mind set more in research nowadays. Bart, thank you for showing that you should push and hustle when you want something. Not to be afraid to send a cold email pitch (and to send it again when a reply doesn't follow). To show how to take on many things at once, feel overwhelmed (just a little), then decide to just bite harder and come out stronger. For not spending too much words or time on useless small talk, dilly-dallying or fine print… But also for spending just enough time whenever it was needed most. I thank you both.

I also wish to extend my gratitude to my family. I thank my parents for their support. Trying to understand some specialized, difficult to relate to topic is not easy, let alone the fact that the "PhD-ing" child is often not in the mood to talk research. But you took it in stride and were there for me, always and unconditionally. Also for my sister Freyja, I can do nothing but simply say thank you. I have no way to express how much I valued our weekend talks. I think of you as my older and wiser sister in the best way possible, even though I am the older sibling purely by chance. Your advice, thoughts, maturity, discipline, kindness and view on life have been immensely inspiring to me. Chris, you have truly been like a brother. I greatly enjoyed talking with you about games (mostly talking), photography (mostly listening), cars (mostly listening as well), or just appreciating your company and hanging out. I hope we will have more opportunities to do so in the future. Lastly, I wish to thank my grandmother, grandfather, aunts, uncles, nephews and nieces. It was always heart-warming to meet.

Even as a PhD "student", you spend most of your time with colleagues, and I am lucky to have found myself surrounded by excellent characters, so much even that it is difficult to figure out where to begin. Perhaps at the beginning: Pieter, thank you for welcoming me in what would later become the "psychiatric-slash-coffee-break-slash-leave-us-alone-we're-working" office. I could not have imagined a better office-mate for the first year or so. I greatly enjoyed our discussions on research, and those dealing with psychological and philosophical (or both?) questions even more. The same can be said for Thomas, another colleague I suffered missing once gone. Thomas, talking with you was always a pleasure, be it about machine learning, human learning, research, physics, little bits of trivia, or life worries. Jochen, you referred to me as being your "research soul mate" in your dissertation; well, I want to repay the favor and state that the same goes for you. From start to finish, you acted as a great mentor and welcoming conversation partner and motivated me to explore my first small research ideas (as a fellow PhD student) as well as providing feedback all the way to my final dissertation (as a committee member)—thank you. Filip, in the beginning, I didn't fully know what to make of you—and I'm still not quite sure now, but what I can say is that you are one of the most excellent, unprejudiced (thankfully so!) and nose-on-the-facts people I ever met. I have to admit I even picked up a thing or two from you (as you so aptly noticed yourself), and I think back with great fondness to our travels together. Aimée, I couldn't do a proper acknowledgment without mentioning you. Whether it was talking in the office (that being: me invading your office for a bit), during our Dodentocht practice walks or during Chinese course, I always found in you a very like-minded friend, let alone fellow China… connaisseur. You're the best "buddy" to have around. I also wish to thank Alex. For being a great travel companion[1], but also for being a great, relaxed, and humorous friend in general. I am also grateful for my wonderful colleagues post-office-move. Johannes, it was a pleasure to work with you as well as to discuss all sorts of topics, ranging from music, programming Java for ProM, computer configurations, or the latest Britney-related musings of fellow researchers. Estefanía, the same goes for you. I really enjoyed getting to know you as a driven, compassionate and friendly colleague. I wish you both the best of luck in your further endeavors. I'd also like to thank my other colleagues throughout the past years: Flavius, Gayane, Helen, Jonas, Karel, Philippe, Libo, Tom, Véronique, Willem and Wouter. I also extend my thanks to all the professors in my department, whom it was a pleasure to work with. Finally, I also wish my best to all the new colleagues having started recently.

I also want to thank some other friends. Though we could only meet rarely, rest assured that I remember those meetings—and friendships—with great fondness. In Leuven, I want to thank Pieter, Flip, Ileen, and Jerome for being such good friends. Not only during my PhD, but also before. Back in Ghent (or what's rapidly becoming Ninove again for most of you), I wish to thank Brecht, Dorien, Evelien, Robin and Tim for all the great times. I am proud to call you all among my friends.

I also wish to express my gratitude for all the people involved in bringing—and welcoming—me to new places. Euripides said: "Experience, travel—these are as education in themselves." and I can do nothing more than agree with this. I therefore wish to thank Prof. Arthur ter Hofstede for welcoming me to Queensland, Australia, as well as acting as an excellent committee member, providing

[1]I hereby grant you the right to become the exclusive authority regarding the Mambo No 5 story retelling[2].

[2]Though kindly don't retell it too often.

not only bright insights regarding process mining and business process management related topics, but also a staggering amount of movie trivia. Without doubt, my stay in Brisbane was worth remembering thanks also to Raffaele, Suriadi, Arthem, Keith, Prof. Moe Wynn and all other friendly people there. Thanks for having me! I also wish to thank Prof. Josep Carmona for welcoming me to Barcelona, Spain, and also for being an upstanding as well as outstanding committee member. I greatly value your support and always constructive feedback, starting all the way from my first seminar. Jorge, thank you as well for being a great companion in Spain. With our noses in the same direction and with similar ways of thinking, we hit it off immediately. I also thank Prof. Guoqing Chen and Prof. Qiang Wei for always providing an excellent welcome in China. I also want to explicitly thank Prof. Chen to act as a committee member and for the great talk and efforts when I was in China a few months ago. I also wish to thank Prof. Jae-Yoon Jung at Kyung Hee University, for the splendid collaboration and the invitation to South Korea. I also wish to thank some other people I had the pleasure to work with and learn from throughout the past years. In particular, I am grateful to Walter Vanherle, Maurizio Queirazza, Peter Beirsmans, Koen de Meulemeester, Geert Smets and François Eyskens.

Finally, it is also an acknowledgment-tradition to end with that one, special person in one's life. Xinwei—my Julie—you are without doubt that person for me. Thank you for all the love, happiness and support you have given me. Thank you for all the amazing, wonderful moments we shared together, and thank you for keeping up with me during my most stressful days. You are the most beautiful woman I know in every sense of the word, and I look forward to continuing our story together. To put it in the clumsy words of a level one Chinese speaker... 我写做一个博士是就像寻找一个明灯。鑫伟，你给我认识这个明灯不是知识和学问的，可是一个爱的。你是我光，我的万事。我爱你一辈子，我们会一直在一起，谢谢你。

– Seppe Vanden Broucke, September 2014

Contents

English Summary

Process mining is the research area that is concerned with knowledge discovery from event logs, which are recorded and stored by a wealth of information systems, used to support, model and govern operational processes. Event logs can contain up to millions of events, and organizations face the challenge of extracting value from this vast data repository, so as to improve their business processes by learning from insights derived from these data sets. Typically, the field of process mining is structured by categorizing its analysis tasks into a taxonomy containing three broad task types: process discovery, conformance checking, and process enhancement.

Although great strides have been made in the field of process mining towards improving business processes and deriving insights from historical event-based data repositories, the field faces some notable difficulties. One particular difficulty is that process mining, with process discovery in particular, is frequently limited to the setting of unsupervised learning, as negative information (i.e. events that were prevented from taking place) are often unavailable in real-life event logs. Multiple solutions have been proposed to resolve this aspect, with one of them encompassing the artificial induction of negative events based on available, positive, information. This concept is put forward as the topic of focus throughout the first part of this thesis. We outline an improved artificial negative event generator, and subsequently enhance this technique with a scoring mechanism to assign a measure of confidence to induced negative events. Next, we show how the induced negative events can be applied in a conformance checking setting, as they allow us to develop a comprehensive conformance checking framework in line with standard machine learning practices, allowing to assess the recall, precision and generalization of a process model. Finally, we also show how artificial negative events can be applied to uncover implementation problems by using them as highlighters of unobserved behavior.

The second part of this thesis outlines a number of additional contributions which are not directly related to the concept of artificial negative events. In particular, a novel heuristic process discovery technique is presented, based on a long lineage of related process discovery algorithms, but with a particular focus on robustness and flexibility. Next, based on replay strategies developed throughout the first part of the thesis, an event-granular conformance analysis technique is presented which can be applied towards enabling real-time monitoring of business activities. Next, a technique for explaining event log cluster solutions on an instance-granular level is presented. Finally, a benchmarking framework is developed to enable the automated set-up of large-scale conformance analysis experiments.

Nederlandse Samenvatting

Process mining behelst het onderzoeksdomein dat zich toelegt op het ontginnen van kennis uit “event logs”, data sets die worden geregistreerd en opgeslagen door informatiesystem die instaan voor het ondersteunen, modeleren en sturen van operationele processen. Event logs kunnen miljoenen events bevatten, en organisaties worden aldus geconfronteerd met de uitdaging om waarde te extraheren uit deze data repositories, met als einddoel het verbeteren van bedrijfsprocessen door te leren uit inzichten, gehaald uit deze data sets. De analysetaken beschreven in het gebied van process mining worden gebruikelijk opgedeeld in de volgende drie categorieën: process discovery (procesontginning), conformance checking (conformiteitsanalyse) en process enhancement (procesverbetering).

Process mining heeft sterk bijgedragen tot het verbeteren van bedrijfsprocessen en het extraheren van inzichten uit historische, event-gebaseerde data sets, maar wordt desalniettemin geconfronteerd met enkele opvallende moeilijkheden. Een welbepaalde uitdaging volgt uit het feit dat process mining, en procesontginning in het bijzonder, vaak gelimiteerd is tot de setting van niet-gesuperviseerde leertaken, aangezien negatieve informatie (events die niet konden of mochten uitgevoerd worden) vaak afwezig zijn in echte, niet-synthetische, event logs. Verschillende oplossingen zijn voorgesteld om dit probleem te verhelpen, waaronder een techniek die zich toelegt op de artificiële inductie van negatieve events gebaseerd op aanwezige, positieve informatie. Dit concept wordt naar voor geschoven als onderwerp voor het eerste deel van deze thesis. We beschrijven een verbeterde artificiële negatieve event generatietechniek, en amplificeren deze vervolgens met een scoresysteem om zo een betrouwbaarheidswaarde te verschaffen aan geïnduceerde negatieve events. Vervolgens illustreren we hoe gegenereerde negatieve events kunnen aangewend worden in een conformiteitsanalyse setting, en werken hiertoe een raamwerk uit in lijn met standaard machine learning praktijken om zo de recall, precisie en generalisatie kwaliteiten van een procesmodel

te evalueren. Ten slotte tonen we aan hoe artificiële negatieve events aangewend kunnen worden om implementatieproblemen op te sporen door ze te gebruiken als aanwijzers van niet-geobserveerd gedrag.

Het tweede deel van deze thesis beschrijft aanvullende contributies die niet rechtstreeks gerelateerd zijn aan het concept van artificiële negatieve events. Een vernieuwende heuristische procesontginning-techniek wordt voorgesteld, voortbouwend op een lange reeks gerelateerde technieken, maar die zich specifiek toelegt op robuustheid en flexibiliteit. Vervolgens wordt er een event-granulaire conformiteitsanalyse techniek voorgedragen, gebaseerd op replay-technieken die ontwikkeld werden in het eerste deel van de thesis, teneinde bedrijfsactiviteiten in real-time op te volgen, gevolgd door een techniek die toelaat om verklaringen af te leiden voor event log cluster-oplossingen op een instantie-granulair niveau. Tot slot werd een benchmarking raamwerk ontwikkeld om de automatische set-up van grootschalige conformiteitsanalyse experimenten mogelijk te maken.

Deutsche Zusammenfassung

Process Mining ist der Forschungsbereich, betroffen mit Wissensentdeckung aus Ereignisprotokollen, die aufgezeichnet und durch eine Fülle von Informationssystemen gespeichert sind, und verwendet werden um operativen Prozesse zu unterstützen, zu modellieren und zu regieren. Ereignisprotokolle können bis zu Millionen von Ereignissen enthalten, und Organisationen stehen vor der Herausforderung der Gewinnung von Wert aus diesem riesigen Daten-Repository, um ihre Geschäfts-prozesse abgeleitet durch das Lernen von Einsichten aus diesen Datensätzen zu verbessern. In der Regel wird der Bereich der Process Mining strukturiert durch die Kategorisierung seiner Analyseaufgaben in einer Taxonomie die drei große Aufgabentypen enthält: Prozesserkennung, Konformitätsprüfung und Prozess-Verbesserung.

Obwohl große Fortschritte im Bereich der Process Mining zur Verbesserung der Geschäftsprozesse und daraus Erkenntnisse aus historischen Ereignis-basierte Daten-Repositories gemacht sind, steht das Feld vor einigen bemerkenswerten Schwierigkeiten. Eine besondere Schwierigkeit ist, daß Process Mining, mit Prozess Entdeckung insbesondere häufig auf die Einstellung des unüberwachten Lernens beschränkt wird, da negative Informationen (dh Ereignisse, die stattfindet verhindert wurden) oft im realen Ereignisprotokolle verfügbar sind. Mehrere Lösungen sind vorgeschlagen worden, diesen Aspekt zu beheben, mit einem von ihnen umfasst die künstliche Induktion von negativen Ereignissen auf der Grundlage verfügbarer, positive Informationen. Dieses Konzept ist zukunfts als Themenschwerpunkt in der gesamten ersten Teil dieser Arbeit. Wir skizzieren eine verbesserte künstliche negatives Ereignis-Generator und anschließend verbessern diese Technik mit einem Scoring-Mechanismus, um ein Maß an Vertrauen induzierten negativen Ereignissen zuzuweisen. Weiter zeigen wir, wie die induzierten negativen Ereignisse in einer Konformitätsprüfung Einstellung angewendet werden, da sie es uns ermöglichen, einen voller Konformitätskontrollrahmen zu entwicklen, in Übereinstimmung mit der üblichen Praxis des maschinellen Lernens, so dass Recall, Präzision und Verallgemeinerung eines Prozessmodells beurteilt worden kann. Schließlich zeigen wir auch, wie künstliche negative Ereignisse angewendet werden, um Probleme bei der Umsetzung, indem sie als Zeiger unbeobachtet Verhalten angewendet werden.

Der zweite Teil der Arbeit beschreibt eine Reihe von Zusatzbeiträge, die nicht direkt auf den Begriff der künstlichen negativen Ereignissen verbunden sind. Insbesondere wird eine neue heuristische Verfahrens Entdeckungstechnik vorgestellt, die basiert ist auf einer langen Linie von verwandten Prozesserkennungsalgorithmen, aber mit einem besonderen Fokus auf Robustheit und Flexibilität. Weiter, auf der in der gesamten ersten Teil der Arbeit entwickelte Replay Strategien basiert, wird ein Ereignis-Konformität granulare Analyse-Technik vorgestellt, die auf die Ermöglichung Echtzeit-Überwachung von Geschäftsaktivitäten angewendet werden können. Als nächstes wird ein Verfahren zum Erläutern Ereignisprotokoll Cluster-Lösungen auf einer Instanz-granulare Ebene dargestellt. Schließlich wurde ein Benchmarking-Rahmen entwickelt, um den automatisierten Aufbau von massenhaften Konformität Ana-lyse-Experimente zu ermöglichen.

中文摘要

流程挖掘(“Process mining”)是一个以事件日志的知识发现为探索重点的研究领域。信息系统记录和存储了大量的事件日志用以支持、模拟和管理操作流程。尽管事件日志能够记录成万上亿的事件，但是组织机构要从如此大的数据资料库中获取价值确实是一个挑战。因此，我们需要对这些数据集进行观察、学习和研究，用以提高和优化业务流程。通常，流程挖掘的研究领域以它对任务分析的分类为依据进行划分，概括为三大类：流程发现(“process discovery”)、一致性检验(“conformance checking”)和流程改进(“process enhancement”)。

尽管流程挖掘在提高业务流程和基于历史日志的数据资料库的研究方面取得了巨大进步，但是仍然面临着一些重大的难题。难题之一，由于负面信息通常在真实的事件日志中不可用，使得流程发现的研究通常局限在非监督学习的情境下进行。为了解决这个问题，研究者提出了多种解决方案，一种是根据可用的正面信息，围绕人工引变负面事件的方法开展。人工负面事件(“artificial negative events”)这个概念作为本研究的主题之一贯穿于论文的第一部分。论文的第一部分首先提出了一种优化的人工负面事件生成器，并且利用评分机制对引变的负面事件进行信心测度的方法来提高这项技术。接下来，这部分展示了引变的人工负面事件如何应用在一致性检验的情境中。它们允许按照标准的机器学习实践方法构建一个全面的一致性检验框架，从而来评价一个流程模型的恢复召回率、精密和一般化。最后，这部分堪称类比人工负面事件为揭未被察觉行为的荧光笔，展示如何应用它们来揭示信息系统实现的有关问题。

论文的第二部分概述了与概念人工负面事件没有直接关联的其他几项成果。首当其冲的是提出了一种全新的启发式流程发现技术。这项技术与的相关的流程发现算法一脉相承，但主要集中在稳健性和灵活性方面。其次，基于论文第一部分的回放的手段(“replay strategies”)，提出了一种事件粒度层的一致性分析技术。这项技术能够用于激活业务活动的实时监控。

再次，在实例粒度层次上，提出了一种解释事件日志集群解决方案的技术。最后，提出了一种能够为开展大规模一致性分析实验进行自动构建的基准框架。

Sommaire Français

L'exploration des processus (process mining) est le domaine de recherche dédié à la découverte de connaissance à partir de journaux d'événements (event logs), qui sont enregistrés et stockés par une multitude de systèmes d'information, et sont utilisés pour soutenir, modéler et gouverner les processus opérationnels. Les journaux d'événements peuvent contenir jusqu'à des millions d'événements, de ce fait les organisations sont ainsi confrontés au défi d'extraire des valeurs de ce vaste entrepôt de données, afin d'améliorer leurs processus d'affaires par l'apprentissage de connaissances issues de ces ensembles de données. En règle générale, le domaine du forage des processus est structuré en classant ses tâches d'analyse selont une taxonomie contenant trois types généraux : la découverte des processus, la vérification de la conformité, et l'amélioration des processus.

Bien que de grands progrès ont été réalisés à l'aide de l'exploration des processus dans l'amélioration des processus d'affaires et dans la dérivation d'un savoir à partir de l'historique des référentiels de données basées sur des événements, le domaine est confrontée à des difficultés notables. Une difficulté particulière est que l'extraction de processus, avec le processus de découverte en particulier, est souvent limitée à la création de l'apprentissage non supervisé, comme l'information négative (c.à.d.des événements qui ont été empêchés d'avoir lieu) sont souvent indisponibles dans les journaux d'événements issus de la vie réelle. De multiples solutions ont été proposées pour résoudre ce problème, avec l'un d'eux entourant l'induction artificielle d'événements négatifs en fonction de la disposition, positive, des informations. Ce concept est présenté comme le sujet de discussion tout au long de la première partie de cette thèse. Nous présentons une amélioration artificielle générateur d'événements négatifs, puis améliorons cette technique avec un mécanisme de notation afin d'attribuer une mesure de confiance pour les événements négatifs induits. Ensuite, nous montrons comment les événements négatifs induits peuvent être appliqués dans un cadre de

contrôle de conformité, car ils nous permettent de développer un cadre de contrôle de conformité complet en ligne avec les pratiques d'apprentissage de la machine standard, afin d'évaluer le rappel, la précision et la généralisation d'un modèle de processus. Enfin, nous montrons aussi comment les événements négatifs artificiels peuvent être appliqués à découvrir les problèmes de mise en œuvre en les utilisant comme des indicateurs de comportement observé.

La deuxième partie de cette thèse présente un certain nombre de contributions supplémentaires qui ne sont pas directement liées à la notion d'événements négatifs artificiels. En particulier, une technique inédite de découverte de processus heuristiques est présentée, basée sur une longue série d'algorithmes de découverte de processus connexes, mais avec un accent particulier sur la robustesse et la flexibilité. Ensuite, sur base de stratégies de relecture développés tout au long de la première partie de la thèse, une technique d'analyse de conformité événement granulaire est présentée qui peut être appliquée en vue de permettre un suivi en temps réel des activités de l'entreprise. Ensuite, une technique pour expliquer les solutions de regroupement (clustering) du journal des événements au niveau de l'instance granulaire est présentée. Enfin, un cadre de référence a été développé pour permettre la mise en place automatique d'expériences d'analyse de conformité en masse.

Outline

"In the beginner's mind there are many possibilities,
but in the expert's there are few."
– Shunryu Suzuki

This dissertation is divided into two parts. Part I outlines an efficient weighted artificial negative event generation technique together with a conformance checking framework wherein such events are applied to assess the quality of a process model. Part II presents contributions which are not directly related to the concept of artificial negative events.

The following outline provides a summary of the works presented in this dissertation.

Part I – Artificial Negative Event Based Techniques for Business Process Conformance Checking

Chapter 1 – Introduction

- This chapter provides a thorough introduction on process mining and outlines the research background and context.
- Background information and related work on the concept of artificial negative events is given.
- Preliminaries and notations used throughout the rest of the dissertation are also provided.

The work presented in this chapter has been published in:

- Dejaeger, K., vanden Broucke, S., Eerola, T., Wehkamp, R., Goedhuys, L., Riis, M., Baesens, B. (2012). Beyond the hype: cloud computing in analytics. Data Insight & Social BI: Executive Update, Cutter Consortium.
- vanden Broucke, S., Baesens, B., Vanthienen, J. (2013). Closing the loop: state of the art in business process analytics. Data Insight & Social BI: Executive Update, Cutter Consortium.
- Baesens, B. (2014). Chapter: "Business Process Analytics", vanden Broucke S. in: Analytics in A Big Data World: The Essential Guide to Data Science and its Applications. Wiley, June 2014.

Chapter 2 – An Improved Artificial Negative Event Induction Technique

- This chapter presents an improved artificial negative event induction technique, extending earlier approaches [1–3].
- The developed technique is a first step towards enabling the generation of a complete and correct artificial negative events in a robust manner and is better able to deal with the problem of event log incompleteness.

The work presented in this chapter has been published in:

- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2012). An improved process event log artificial negative event generator. FEB Research Report KBI_1216. Leuven (Belgium): KU Leuven - Faculty of Economics and Business.
- vanden Broucke, S., De Weerdt, J., Baesens, B., Vanthienen, J. (2012). An improved artificial negative event generator to enhance process event logs. In Lecture Notes in Computer Science. International Conference on Advanced Information Systems Engineering (CAiSE'12). Gdansk (Poland), 25-29 June 2012 (pp. 254-269) Springer.

Chapter 3 – Determining Process Model Precision and Generalization with Weighted Artificial Negative Events

- This chapter introduces the concept of weighted artificial negative events.
- The generation technique presented in the previous chapter is extended and optimized for use in a conformance checking context.
- An empirical evaluation shows the validity of the scoring metric used towards assigning a weighting to artificial negative events.
- Based on the concept of weighted artificial negative events, a comprehensive conformance checking framework is presented, which is able to assess the recall, precision and generalization quality dimensions of process models in a robust, integrated, and scalable manner.
- We develop a series of replay strategies and negative event evaluation methods to implement the conformance checking framework.
- We present a benchmarking study to compare our developed metrics and techniques with other conformance checking approaches.

The work presented in this chapter has been published in:

- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2013). Determining process model precision and generalization with weighted artificial negative events. IEEE Transactions on Knowledge and Data Engineering.
- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2013). On replaying process execution traces containing positive and negative events. FEB Research Report KBI_1311. Leuven (Belgium): KU Leuven - Faculty of Economics and Business.

Chapter 4 – Artificial Negative Events as Unobserved Events

- The final chapter in this part applies the concept of artificial negative events as a detection mechanism for unobserved behavior.
- We show how this can help to highlight implementation problems, both for procedural and declarative process models.

- This idea forms the basis of a more comprehensive event existence classification framework.

The work presented in this chapter has been published in:

- Caron, F., vanden Broucke, S., Vanthienen, J., Baesens, B. (2012). On the distinction between truthful, invisible, false and unobserved events. Sprouts: Working Papers on Information Systems: vol. 12 (16). 11th JAIS Theory Development Workshop at ICIS 2012. Orlando, Florida, 16 December 2012.
- Caron, F., vanden Broucke, S., Vanthienen, J., Baesens, B. (2012). On the distinction between truthful, invisible, false and unobserved events. Proceedings of the 18th Americas Conference on Information Systems (ACIS 2012): Vol. e-pub. Americas Conference on Information Systems. Seattle, Washington (US), 9-12 August 2012 (art.nr. 24) Association for Information Systems.
- vanden Broucke, S., Caron, F., Vanthienen, J., Baesens, B. (2013). Validating and enhancing declarative business process models based on allowed and non-occurring past behavior. Business Process Management Workshops. Workshop on Decision Mining & Modeling for Business Processes (DeMiMoP'13). Beijing (China), 26-30 August 2013.
- vanden Broucke, S., Caron, F., Lismont, J., Vanthienen, J., Baesens, B. (2014). On the Gap between Reality and Registration: A Business Event Analysis Classification Framework, *in submission (Journal of Information Technology & Management, Springer).*

Part II – Other Advances in Process Mining

Chapter 5 – Fodina: Robust and Flexible Heuristic Process Discovery

- A novel heuristic process discovery technique, called Fodina, is presented.
- The technique is based on a long lineage of related process discovery algorithms, but with a particular focus on robustness and flexibility.
- The technique is able to discover the construct of duplicate activities.

- A benchmarking study confirms the robustness and scalability of the technique for both synthetic and real-life event logs.

The work presented in this chapter has been published in:

- vanden Broucke, S., De Weerdt, J., Vanthienen J., Baesens, B. (2014). Fodina: a Robust and Flexible Heuristic Process Discovery Technique, *in submission (Information Systems, Elsevier).*
- De Weerdt, J., vanden Broucke, S., Caron, F. (2014). Bidimensional Process Discovery for Mining BPMN Models. Workshop on Decision Mining & Modeling for Business Processes (DeMiMoP'14). Haifa (Israel), 7-11 September 2014, accepted.

Chapter 6 – Event-Granular Real-Time Decomposed Conformance Analysis

- Based on the replay strategies developed throughout the first part of the thesis, an event-granular conformance analysis technique is presented.
- The technique can be applied towards enabling real-time monitoring of business activities.
- By applying the concept of process model decomposition, the technique is able to localize deviations in a more precise and fine-grained manner.
- The technique can be ran in a distributed manner.

The work presented in this chapter has been published in:

- vanden Broucke, S., Muñoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J. (2013). Event-based real-time decomposed conformance analysis, 21 pp: Polytechnic University of Catalonia, Department of Information Languages and Systems.
- vanden Broucke, S., Muñoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J. (2014). Event-based real-time decomposed conformance analysis, 18 pp. OnTheMove Federated Conferences & Workshops, CoopIS 2014 (CoopIS'14), Amantea, Calabria (Italy), 27-31 October 2014.

Chapter 7 – Explaining Clustered Process Instances

- This chapter presents SECPI (Search for Explanations for Clustered Process Instances), a technique for explaining event log cluster solutions on an instance-granular level.

The work presented in this chapter has been published in:

- De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Explaining Clustered Process Instances, Business Process Management Conference 2014, Haifa (Israel), 7-11 September 2014.

Chapter 8 – A Conformance Analysis Benchmarking Framework

- This chapter presented CoBeFra (the Comphrensive Benchmarking Framework): a technique devoted to enable the automated set-up of mass-scale conformance analysis experiments.
- We present a developed Petri net based event log generation technique which enables to rapidly construct a set of synthetic event logs for use within experimental setups.
- We present the results of a benchmarking study aiming to uncover the relationship between process discovery techniques and event log characteristics.

The work presented in this chapter has been published in:

- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2013). A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM. Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, SSCI 2013. Singapore, 16-19 April 2013.
- vanden Broucke, S., Delvaux, C., Freitas, J., Rogova, T., Vanthienen, J., Baesens, B. (2013). Uncovering the relationship between event log characteristics and process discovery techniques. BPM Workshops (BPI2013). Beijing (China), 26-30 August 2013.

- vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Straightforward Petri Net-based Event Log Generation in ProM. FEB Research Report, Leuven (Belgium): KU Leuven - Faculty of Economics and Business.

Part I

Artificial Negative Event Based Techniques for Business Process Conformance Checking

Chapter 1

Introduction

"Things always become obvious after the fact."
– Nassim Nicholas Taleb

This chapter provides introductory background material regarding process mining, as well as providing preliminary notations and definitions. In addition, the research context and background regarding the concept of artificial negative events is introduced. This chapter can optionally be skipped by readers who are already familiar with the field.

1.1 Process Mining

1.1.1 An Avalanche of Data

Many of today's organizations are currently confronted with a true avalanche of data. Especially concerning business processes, i.e. the area of interest in this dissertation, an incredible amount of investments has been made throughout the past decades, subsequently driving an incredible growth in the availability of process-related data, i.e. operational transaction logs, execution traces of business activities, and so on. Given this fact, it is without doubt that important opportunities arise for analysis of process-related data in order to provide insights into the actual way of working, monitoring business activities, detecting deviations or improving performance criteria. However, performing such analysis tasks is oftentimes easier said than done. Although information support systems such as Enterprise Resource Planners (ERP) and modern Workflow Management Systems (WfMS) commonly provide analysis and visualization tools in order to monitor and inspect processes—often based on key performance indicators—an abundance of data about the way people conduct day-to-day practices still remains untapped and concealed inside process-related data repositories.

The difficulty towards analyzing these data repositories stems from a number of key characteristics. First, because of the nature of IT related evolutions, the possibilities towards storing data are ever increasing (decreasing cost of storage). This leads to the creation of vast data repositories, archived "for later use", which contain a wealth of potential knowledge-based value, but are impossible to analyze in a manual manner. When enterprises succeed in applying advanced analysis techniques on such data sets, the practice is frequently denoted as "big data analytics", whereas companies that don't succeed in doing so face an "information overload". In short, *big data sets require appropriate techniques.* Second, somewhat paradoxical, an increase in automation oftentimes causes an increase in complexity. The problem stems from the fact that although automation and information technology allow to integrate many departments across the organization (which manager hasn't dreamed of the ultimate, encompassing workflow model?), it becomes harder and harder for analysts, process owners and other stake holders to retain an overview of the overall process "inside their heads", that is, *increasing automation comes with a complexity cost.* The third and final key difficulty is due to the "gaps" which exist between the design, implementation, and execution of business processes. The as-designed process model might not be the one which end up being implemented in the innards of some information system. Sometimes this is due to human error (a programming mistake), due to the information system or tooling being used (your workflow system uses the X modeling language while your designers model with Y), or simply due to a conscious decision (the designed model is more like a prescriptive guideline and should not always be followed to the limit). Next, the as-implemented process might not even be in correspondence with the as-executed process. Perhaps the system allows to skip some tasks. Perhaps sometimes this is even desired, perhaps some process instance had to be restarted, or prematurely canceled. Perhaps there is still some three-year running instance open in the system. In summary, there is *a gap between the was-designed, as-is and to-be process.*

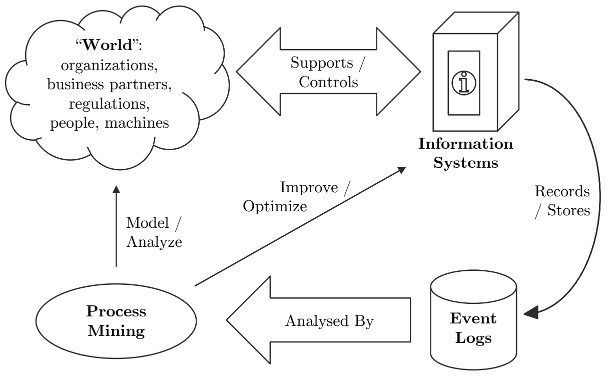


Figure 1.1: Process mining models and analyzes the (organizational) context in which it is applied, supported and controlled by information systems, which process mining hence tries to improve and optimize. To do so, process mining starts from process-related data as stored in so-called event logs, which are recorded and stored by aforementioned information systems. (Adapted from [4].)

1.1.2 Knowledge Discovery from Event Logs

The size of process-related data repositories, together with their increasing complexity and the realization that there exists a gap between process model design, implementation and execution has caused practitioners and scholars to realize that strong, robust analysis techniques should be pursued to obtain valuable insights regarding business processes. As such, throughout the past years, a research field has sprung up which aims to extract knowledge discovery from event logs; that is, process-related data sets [4]. The field is denoted as *Process Mining* and is oftentimes situated at the intersection of the fields of data mining [5, 6], because of the shared goal of learning from data repositories and the field of Business Process Management (BPM), as its major objective entails gaining insight into business operations, so that process mining fits within the diagnosis phase of the BPM life cycle [7]. Figure 1.1 shows the typical, well-known schematic which situates process mining in its organizational context. Process mining models and analyzes this context, which is supported and controlled by various sorts of information systems. As such another aim of process mining is to improve and optimize these information systems. To do so, process mining techniques starts from process-related data as stored in so-called event logs, which are recorded and stored by aforementioned information systems.

Typically, the field of process mining has been structured by categorizing all analysis tasks and techniques into a taxonomy containing three broad task types

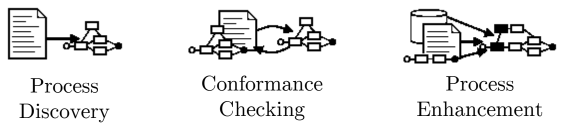


Figure 1.2: Process mining task are frequently categorized into three broad types: process discovery, conformance checking and process enhancement.

(see Figure 1.2). The first task type, denoted as *process discovery*, deals with the construction of a process model out of an event log. In most cases, no a-priori model or knowledge is assumed, although some—though few—techniques are able to take into account domain-knowledge to derive models as well. This task type kick started the field of process mining around a decade ago and has as such received the most attention in the research community (in some cases or sources, the term process mining is used interchangeably albeit confusingly with process discovery). In most cases, the model being discovered is represented in a well-known process modeling language, such as BPMN or other procedural-oriented notations like Petri nets, but depending on the log perspective being investigated and the goals of the analysts, other models, such as social networks or declarative models, can be extracted as well. The second task type, called *conformance checking* (or: *conformance analysis*) starts from an event log and a process model and analyzes the quality of the process model based in terms of its ability to represent the behavior captured in the event log (how well is the model able to represent reality) or, alternatively, attempts to explain where deviations have occurred during the execution of the process (where did reality violate or deviate from the prescribed model). The last task, *process enhancement* (or: process extension), also assumes the presence of an existing process model and tries to improve, enrich or extend this model based on additional or more recent data. In many cases, "traditional" data mining practices also find their way in many of the analysis tasks process mining encompasses, for example to uncover the root-causes behind deviations in the form of a decision tree, or to build predictive models to estimate how much time a process instance will consume before completion, for instance.

1.1.3 The Event Log as the Center Point of Analysis

As mentioned before, the starting point for each process mining analysis task is a so-called event log, a repository of process-related data, transactional records, or traces of operational behavior. Most process mining techniques can be applied on any sort of event log, but a basic number of requirements must be met. That is, at the very least, entries in the event log (i.e. the actual events) should contain:

- A *case identifier*, or instance identifier, which serves to group multiple events together, indicating that they relate to the same process instance, the same case.
- Events must be *labeled*. That is, they must carry some name. In most cases, events are labeled according to the activity they represent, for instance "sign contract". Typically, these labels will be used to fill in the activity "boxes" in a discovered process model. A common "trick of the trade" which is used in practice is to vary the manner by which events are labeled. For instance, a "handover-of-work" network can be derived by using the department who has executed a specific activity as the label, rather than the actual activity name itself.
- Finally, events must exhibit a form of *ordering*. In the most optimal case, all the events in the event log contain a starting and ending timestamp denoting the exact point in time the event was started and completed, allowing to order all events and all process instances. In most real-life event logs, however, only one atomic timestamp is provided for each event. When no absolute timestamp information is available, techniques are also able to work with relative ordered events (a sequential row number can suffice), either over the whole event log or only for the events in each process instance (this was the first event in the instance, followed by the second, etc.). In the last case, it is not possible to establish an ordering between process instances, but this can still serve as sufficient input for many process mining techniques.

An important aspect to note is that the requirements of a case identifier and an ordering cause an event log are different from a normal table-based, flat data set as used in "traditional" machine learning. That is, an event log works on two levels of aggregation: events, and a grouping of events (a process instance, also frequently denoted as a *trace* of execution). Second, there exist a temporal relationship between events.

Apart from these three basic requirements, event logs can contain a wealth of additional attributes, which can be of help during the execution of process discovery, conformance checking or process enhancement tasks. In fact, when event log provides a rich source of data, analysts can decide upon various log perspectives to adopt to drive the execution of a process mining technique. We already hinted at using the department as an activity label, thus changing the control-flow perspective to a social, organizational one, but other perspectives can be adopted as well, such as:

- Control-flow/activity perspective: focuses on the activities and their ordering of activities in the event log, i.e. the flow of execution between them. This is the most common perspective.
- Social/organizational/resource perspective: focuses on the people, employees, or departments who perform the activities.
- Time/performance perspective: this perspective focuses on the time and performance aspect, for instance to uncover bottlenecks in the process, activities which were cycled or repeated many times or activities which, on average, take a long time to complete.
- Exception perspective: focuses specifically on problems and failures: e.g. prematurely ended traces, noise, missing data, and so on.
- Case data perspective: all other extra data elements and attributes that might be available in the event log, either relating to a specific event, a trace, or the event log as a whole.

It should be said that there exists some overlap between the various perspectives. For instance, when the goal is to detect long-running activities in the context of service level agreement (SLA) violations, this can be regarded as fitting in either the time or exception perspectives. The event log perspective taxonomy is not as strict as the three categories of process mining techniques (various sources also drop some perspectives or name them differently), but an important remark to make is the orthogonal nature amongst these two categories. That is, within the field, it is common practice to pinpoint a specific technique or class of techniques by describing both the process mining analysis task is relates to and the event log perspective it focuses on. Process discovery combined with the control-flow perspective, for instance, leads to a set of discovery techniques which aim to discover a Petri net model, for example (this has been the most popular task-perspective pair in the field so far). Conformance checking combined with control-flow then leads to replay or alignment techniques. Process discovery combined with the social perspective leads to social network discovery. Process enhancement with the time or performance perspective leads to visual annotation techniques where parts of the process model are highlighted based on how much they constrain the overall efficiency, and so on. It is not always straightforward to place a specific technique in this matrix, and in many cases, various techniques are combined and mixed and matched to reach the objective of the analysis.

1.1.4 Crossing Boundaries

1.1.4.1 Process Mining and Business Process Management

Earlier on, we have stated that process mining is situated between the boundaries of BPM and traditional data mining. Figure 1.3 (adapted from [8]) provides a schematic overview on how process mining aims to bridge the gap between existing process related and data oriented techniques to enable performance and compliance oriented analysis exercises.

As process mining concerns itself with improving business processes, the overlap with BPM is relatively obvious. BPM entails the holistic management approach including concepts, methods and techniques to support the design, configuration, enactment, and analysis of business processes [7]. Without doubt, one of the main parts of BPM encompasses process modeling, which refers to the identification, specification, and structured design of business processes. Important phases can be identified in the life cycle of a business process, such as design, modeling, implementation, execution and diagnosis [7], see Figure 1.4. Consequently, process mining fits neatly in the “diagnosis/optimization”-phase of the BPM life cycle. While traditional approaches start out by planning, designing and modeling a process, process mining starts from the processes that are already in place, i.e. from recorded data.

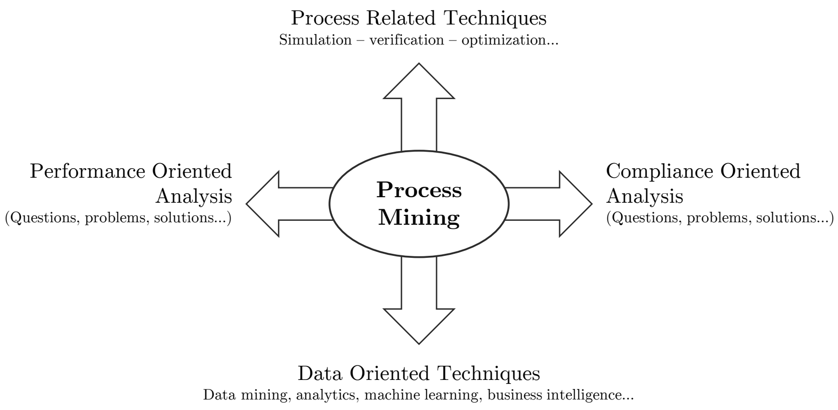

Process Related Techniques
Simulation – verification – optimization...
Performance Oriented Analysis
(Questions, problems, solutions...)
Process Mining
Compliance Oriented Analysis
(Questions, problems, solutions...)
Data Oriented Techniques
Data mining, analytics, machine learning, business intelligence...

Figure 1.3: Process mining on the boundary between process related and data oriented techniques: connecting BPM, simulation, operations research, data mining, and others. (Figure adapted from [8].)

1.1.4.2 Process Mining and Quality Management

BPM is oftentimes described as a "process optimization methodology" and hence mentioned together with related quality control terms such as Total Quality Management (TQM), Six Sigma, or Continuous Process Improvement (CPI) methodologies [9–12]. However, this description is somewhat lacking. Indeed, one significant focal point of BPM is the actual improvement and optimization of processes, but the concept also encompasses best practices towards the design and modeling of business processes, monitoring (consider for instance compliance requirements), and gaining insights by unleashing analytical tools on recorded business activities.

The same holds when comparing process mining with existing quality management related approaches. Process mining can help as an environment-agnostic tool to aid in the reduction of costs and the improvement of quality. Some key differences exist between process mining techniques and existing quality management techniques. Process mining techniques do not assume an a-priori model, whereas many traditional quality control techniques do assume that there exists at least some version of a model to start from. Second, in many quality management techniques, the "activity is the process". That is, oftentimes, the focal point

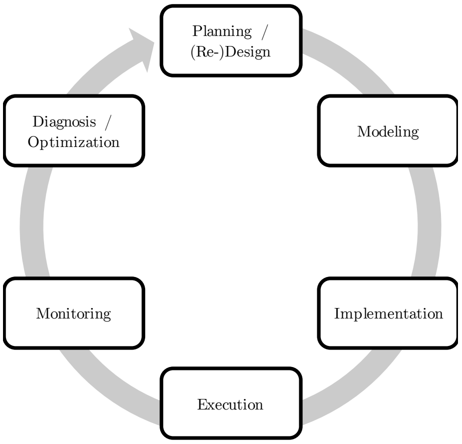


Figure 1.4: The BPM life cycle [7].

of analysis when using statistical process control (SPC) techniques corresponds with a single activity in the encompassing business chain, whereas process mining techniques are able to consider the complete business process as a whole.

1.1.4.3 Process Mining and Business (Process) Intelligence

Just as process mining fits neatly within the diagnosis phase of the BPM life cycle, other domains, such as business (process) intelligence or analytics (BI/BA) also fit in this phase. “Process intelligence” has evolved into a very broad term encompassing a multitude of tools and techniques, and hence nowadays can include anything that provides information to support decision-making [13–18]. As such, in similar fashion as occurred with traditional (“flat”, or tabular) data oriented tools, many vendors and consultants have defined process intelligence to be synonymous with process-aware query and reporting tools, oftentimes combined with straightforward visualizations in order to present aggregated overviews on business actions. In many cases, a particular system will present itself as being a helpful tool towards process evaluation by providing key performance indicator (KPI) dashboards and scorecards, thus presenting a “health report” for a particular business process. In some cases, such tools are presented as monitoring and improvement tools as well (as such covering another phase in the BPM lifecycle), especially when they come equipped with realtime capabilities, compared to batch reporting where the gathering and preparation of reports follows after the actual process execution. Many process-aware information support systems also provide online analytical processing (OLAP) tools to view multidimensional data from different angles and to drill down into detailed information. Another term which has become commonplace in a process intelligence context is business activity monitoring (BAM), which refers to real-time monitoring of business processes, thus related to the real-time dashboards mentioned above, but oftentimes also reacting immediately if a process displays a particular pattern. Corporate performance management (CPM) or business operations management (BOM) are other buzzwords for measuring the performance of a process or the organization as a whole.

Many of the tools and three-letter acronym techniques mentioned above have to deal with similar challenges as process mining has to deal with when analyzing processes, e.g. collecting data and preparing it for use. Although they all provide valid ways to measure and query many aspects of a business' activities, most tools unfortunately suffer from the problem that they are unable to provide real insights or uncover meaningful, newly emerging patterns. Just as for non-process related data sets, although reporting, querying, inspecting dashboard indicators, aggregating, slicing, dicing and drilling are all perfectly valid and reasonable tools for operational day-to-day management, they all have little to do with real process analytics—the emphasis being placed on "analytics". The main issue lies in the fact that such tools inherently assume that users and analysts already know what to look for, as writing queries to derive indicators assumes that one already knows these indicators of interest. As such, patterns that can only be detected by applying real analytical approaches remain hidden. For instance, a KPI dashboard is able to provide feedback regarding some important statistic or metric, but does not show how the actual process is working, comparable to a doctor saying that a patient's organ is ill, without being able to diagnose the actual illness—and, hence, provide a cure. Whenever a report or indicator does signal a problem, users often face the issue of then having to go on a scavenger hunt in order to pinpoint the real root cause behind the problem, working all the way down starting from a high-level aggregation towards the source data. In those cases, process mining is a complementary tool that allows to analyze the processes and find out the root causes of problems, or analyze processes in an exploratory, open-minded manner.

1.1.4.4 Process Mining and Data Mining

The field of process mining also strongly relates to that of data mining (also called: data analytics, business analytics, machine learning, knowledge discovery in databases). The latter is often defined as the non-trivial process of identifying valid, novel, useful, and understandable, explainable patterns in data [5, 6, 19–28]. Data mining is older than process mining, but rarely focuses on processes. The main difference lies in the fact that traditional data mining techniques focus on "flat", data sets, whereas an event log works on two levels of aggregation: events and traces. As such, a gap exists between these two categories of tools, causing difficulties when filtering, exploring and analyzing event based data sets. For example, since event logs are often handled as a large data table (i.e. a listing of events), it is difficult to derive statistical information at the trace level (i.e. a grouping of events) rather than at the event level. Filtering out events based on complex control-flow based criteria is even more challenging. Vice versa, tools which are specifically geared towards the analysis of process based data are often lacking in terms of filtering or descriptive statistics features.

Due to the mismatch described above, practitioners and researchers currently often find themselves using an abundance of tools to perform their analysis tasks. Many possibilities exist to combine the two areas, as they are largely complementary, and indeed we can observe an increase in research attention being devoted to develop techniques which combine the two [1, 29–32].

A note can be made here regarding the aspect of "big data"—i.e. the challenge of extracting value out of gigantic data repositories. At first sight, process mining does not really seem to concern itself about big data, as most big data examples are unstructured and truly enormous, while most process mining techniques do not assume terabytes of data. On the other hand, throughout the last years, we have observed an increase in process-related data, which can all be used in process mining exercises. That said, many process mining techniques are not quite yet ready for "prime time" concerning the analysis of large data sets. Both the development of scalable techniques and the development of approach which can be ran in a parallel manner remains one of the bigger challenges in process mining today.

1.1.4.5 Process Mining and Simulation

In a nutshell, process mining generally aims to generate models in order to understand the current as-is behavior, whereas simulation techniques try to predict future behavior based on provided models. In that manner, one can say that process mining is, in fact, the opposite or counterpart of simulation. Simulation can nonetheless greatly benefit from process mining as the latter can be used as a tool to deliver the parameters needed to configure a simulation model (execution times, waiting times, utilization levels, distribution of arriving cases, or even a complete control-flow model) based on real-life information. This way, process mining can help to build more accurate and better simulation models [33–37].

1.2 Preliminaries

This section provides preliminary definitions which will be applied throughout the remainder of this dissertation, as well as an overview regarding the current state of the art on process discovery and conformance checking techniques.

1.2.1 Definitions

1.2.1.1 Event Logs

An event log consists of events that pertain to process instances: executions of a process in a system. In order to obtain a usable event log, it is assumed that it is possible to record events so that each event is labeled, the event refers to a process instance and that the events are ordered, either based on a time stamp or on the basis of relative ordering (a sequence number).

Definition 1.1. Event. An event is defined as a tuple $e = (c, t, a_1, \dots, a_n)$ with c the case identifier, t the timestamp and $a_1, \dots, a_n$ an arbitrary number of additional attributes. □

Next, we need to establish a way to assign labels to events. We do so through the concept of a classifier function.

Definition 1.2. Classifier. A classifier is a labeling function which maps an event to an unambiguous label, i.e. a hash derived from one or more attributes of the event. □

In most cases, events are classified simply based on the “activity” attribute (a text field describing which activity was performed). We can thus define the following functions: $\mathit{Case} : e \mapsto c$ returns the case identifier of an event, $\mathit{Label} : e \mapsto l$

returns the label for an event (this is the classifier) and $\mathtt{Time} : e \mapsto t$ returns the timestamp of an event.

When activities in a process consume some amount of time (i.e. they are not atomic), multiple ways exist to encode this as an event or a set of events. In some cases, the ending time is encoded directly in the event itself, i.e. $e = (1, \text{"}\mathtt{check\ documents}\text{"}, 5, 7)$. An alternative method is to regard an event as an encoding of a state change of an activity, thus leading to a start event $e_s = (1, \text{"}\mathtt{check\ documents}\text{"}, 5, \text{"}\mathtt{started}\text{"})$ and a completion event $e_e = (1, \text{"}\mathtt{check\ documents}\text{"}, 7, \text{"}\mathtt{completed}\text{"})$. The latter is useful when more than two state changes (i.e. cancellation, restart) need to be logged[1].

Definition 1.3. Trace. A trace σ is a finite sequence of events $\sigma = \langle \sigma_1, \ldots, \sigma_{|\sigma|} \rangle$ with $|\sigma|$ the length/size of the trace and $\sigma_i \in \sigma$ the event at position i in trace σ. It holds that $\forall \sigma_i, \sigma_j \in \sigma, i < j : [\mathtt{Time}(\sigma_i) < \mathtt{Time}(\sigma_j) \wedge \mathtt{Case}(\sigma_i) = \mathtt{Case}(\sigma_j)]$, i.e. the trace is ordered and a grouping of events based on the case identifier. □

Finally, an event log can be defined as a grouping of traces.

Definition 1.4. Event log. An event log L is defined as a multiset of traces, with $|L|$ the cardinality/size of an event log, denoting the total number of traces in the log. It holds that $\forall \sigma \in L : [\nexists \tau \in L, \tau \neq \sigma : [\mathtt{Case}(\sigma) = \mathtt{Case}(\tau)]]$. □

Note the use of function $\mathtt{Case} : \sigma \mapsto \mathtt{Case}(\sigma_1)$ returning the case identifier of the first event in the trace, and thus by definition of the whole trace (all events in the trace). Similarly, we define $\mathtt{Time} : \sigma \mapsto \mathtt{Time}(\sigma_1)$ and $\mathtt{Label} : \sigma \mapsto \langle \mathtt{Label}(\sigma_i), \ldots, \mathtt{Label}(\sigma_{|\sigma|}) \rangle$. We remark that the latter notation will oftentimes be used as a *direct representation of a trace*, leaving the case identifier and exact timestamps implicit. This leads to the definition of a simplified event log.

Definition 1.5. Simplified event log. An event log L is defined as a multiset of traces, with $|L|$ the cardinality/size of an event log, denoting the total number of traces in the log. A trace σ is a finite sequence of labels $\langle \sigma_1, \ldots, \sigma_{|\sigma|} \rangle$, which will also be referred to as "activities" or "events". Two traces are equal, $\sigma = \tau \iff$

[1] A note regarding the event classifier (or labeling function): as we have indicated, in most cases, the label, or: class, of an event is simply equal to the activity attribute of the event. In other cases, as was also mentioned earlier, another attribute, such as the department having executed the activity is used to classify events. In yet some other cases, the activity attribute together with the state change ("start", "complete"...) is used to classify events (to differentiate between events denoting the start and completion of an activity respectively). In very rare cases, it can be interesting to also use the case identifier as an input to construct event classes. An in-depth discussion of why and where this is useful is out of scope of this discussion.

$|\sigma| = |\tau| \land \forall i \in \{1, \ldots, |\sigma|\} : [\sigma_i = \tau_i]$. Two event logs are equal, $L_1 = L_2 \iff \forall \sigma \in L_1 \cup L_2 : [|\{\tau \in L_1 | \sigma = \tau\}| = |\{\tau \in L_2 | \sigma = \tau\}|]$. The number of times a trace appears in the simplified event log L is then called the *multiplicity* of σ. □

An example can help to illustrate this. Consider the event log L containing a single trace $\sigma = \langle \mathtt{start}, \mathtt{register}, \mathtt{participate}, \mathtt{getResult}, \mathtt{complete} \rangle$ with $\mathtt{start} \in \sigma$ instead of writing $\exists x \in \sigma : [\mathtt{Label}(x) = \mathtt{start}]$ and $\sigma_i = \mathtt{start}$ instead of $\mathtt{Label}(\sigma_i) = \mathtt{start}$. Unless indicated otherwise, we will use the simplified event log as a formalization for event logs, traces and events throughout this dissertation for the sake of brevity. Note that a non-simplified event log can easily be converted to a simplified one based on any given labeling function $\mathtt{Label} : e \mapsto l$.

Definition 1.6. Log alphabet. Let A_L be the alphabet over an event log L, so that $A_L = \{\sigma_i \in \sigma | \sigma \in L\}$, i.e. the set of all event classes/labels appearing in the event log. □

For simplified event logs, it is also a frequent practice to work with a "grouped event log" or "distinct event log", which is simply the set ot traces contained in the event log rather than the multiset.

Definition 1.7. Distinct event log. Let $\bigcup L$ be the distinct event log defined over the simplified event log L, i.e. the set $\{\sigma \in L\}$. $|\bigcup L|$ is the distinct cardinality or the number of unique traces (in terms of its sequence of event labels) in event log L. □

An example can help to illustrate this. Consider the event log $L = \{\langle a, b, c, d\rangle^3, \langle a, c, b, d\rangle^2, \langle a, b, d\rangle^4\}$. We use the labeling function to write down traces in a direct manner. $|L| = 3 + 2 + 4 = 9$. $\bigcup L = \{\langle a, b, c, d\rangle, \langle a, c, b, d\rangle, \langle a, b, d\rangle\}$ and $|\bigcup L| = 3$. The multiplicity of $\langle a, b, d\rangle$ equals 4.

1.2.1.2 Process Models

Throughout this dissertation, Petri nets will frequently be applied as the representational language for process models. Petri nets provide a graphical, formal language to represent, analyze, verify and simulate dynamic behavior [38, 39]. A Petri net is a directed bipartite graph, with nodes either representing transitions or places.

Definition 1.8. Petri net. A Petri net is a triple (P, T, F) with: P a finite set of places; T a finite set of transitions with $(P \cap T = \emptyset)$; $F \subseteq (P \times T) \cup (T \times P)$ a finite set of directed arcs (flow relation). $\square$

A place (drawn as a circle) $p \in P$ is called an input/output place of a transition (drawn as boxes) $t \in T$ if there exists an arc from p to t or from t to p respectively. $\bullet t$, $t\bullet$, denotes the set of input/output places for a transition $t \in T$. Similarly, $\bullet p$, $p\bullet$ define the set of transitions having $p \in P$ as an input/output place. That is, let $\bullet x = \{y | (y, x) \in F\}$ and $x\bullet = \{y | (x, y) \in F\}$ with $x, y \in P \cup T$.

Definition 1.9. Marking, marked Petri net. A marked Petri net is a triple (N, M, M_0) with: $N = (P, T, F)$ a Petri net; $M : P \rightarrow \mathbb{N}_0^+$ is a marking function; $M_0 : P \rightarrow \mathbb{N}_0^+$ is the initial marking function. $\square$

Places $p \in P$ in a Petri net can contain zero or more "tokens" (drawn as black dots). The distribution of tokens over the places defines the state, denoted as the "marking" of the Petri net, i.e. the marking function M, which maps each $p \in P$ to a natural, positive number, representing the amount of tokens contained in that place. The multiplicity of a place p in a marking M, i.e. $M(p)$, denotes the number of tokens that this place contains. The initial marking M_0 initializing all places with an initial token count.

Definition 1.10. Petri net execution semantics. The number of tokens may change during the execution of a Petri net. The marking of a Petri net defines a state, based on which execution semantics can be defined. A transition $t \in T$ is said to be enabled under marking M iff each of its input places contains at least one token, i.e. $\forall p \in \bullet t : [M(p) > 0]$. An enabled transition $t \in T$ can be fired, which brings the net from one state to another: $M_1 \xrightarrow{t} M_2$ so that:

$$\forall p \in P : [\begin{cases} M_2(p) = M_1(p) & \text{iff } p \notin \bullet t \cup t\bullet \\ M_2(p) = M_1(p) - 1 & \text{iff } p \in \bullet t \\ M_2(p) = M_1(p) + 1 & \text{iff } p \in t\bullet \end{cases}]. \quad \square$$

Scholars modeling control-flow dimensions of a business process often utilize a subclass of Petri nets, called WorkFlow nets (or, WF-nets) [40–42]. A WF-net specifies the behavior of a single process instance in isolation.

Definition 1.11. WF-net. A Petri net (P,T,F) is a WF-net iff: there is a single source place $i \in P$ such that $\bullet i = \emptyset$; there is a single sink place $o \in P$ such that $o\bullet = \emptyset$; the net $(P, T \cup \{t'\}, F \cup \{(o, t'), (t', i)\})$ is strongly connected, i.e. every $x \in P \cup T$ lies on a path from i to o. $\square$

For conformance checking purposes, we need to establish the concept of trace replay. First, a mapping between transitions in a Petri net and labels in an event log need to be defined.

Definition 1.12. Transition-activity mapping. Let $\mu : T \mapsto A_L \cup \{a_i, a_b\}$ be a function mapping transitions to either a label (thus: an activity in most cases) contained in A_L (the log alphabet of event log L), to an unobservable activity a_i, or to an inaccessible activity a_b. Each label contained in A_L can hence be associated with one or more transitions. It follows that μ assigns a label to each transition in a Petri net. Transitions mapped to a_i are labeled as invisible (silent) transitions (their firing does not lead to an event). Multiple transitions which are mapped to the same activity are duplicate transitions. Executing transitions which are mapped to a_b (a dummy activity outside those contained in A_L) is prohibited. □

Based on the transition-activity mapping μ, it is possible to replay a trace σ on a marked Petri net. The problem of trace replay can be summarized as follows: given a certain trace σ, can a valid sequence of transitions be found in a Petri net such that applying mapping μ on this sequence results in the same trace?

Definition 1.13. Petri net trace replay. In a marked Petri net, enabled transitions can be fired normally. When a transition is fired which is not enabled, we say that the transition is force fired. Force firing a transition $t \in T$ consumes a token from each of its input places $p_{in} \in \bullet t$ where $M(p_{in}) > 0$ and produces a token in each of its output places $p_{out} \in t\bullet$. A firing sequence $f^{M_0,(P,T,F)}$ is a finite sequence of transitions $\langle t_1, t_2, \ldots, t_n \rangle \in T$, with M_0 the initial marking and M_i ($0 < i \leqslant n$) the resulting marking after (force) firing t_1 up to and including t_i sequentially (the firing sequence and Petri net are left implicit), i.e. under $M_0 \xrightarrow{t_1} M_1 \ldots \xrightarrow{t_n} M_n$. A trace $\sigma \in L$ can be replayed by the Petri net (P, T, F) if a firing sequence $f^{M_0,(P,T,F)}$ can be found such that $\sigma_i = \mu(f_i^{M_0,(P,T,F)*})$ with $0 < i \leqslant |\sigma|$ and $f^{M_0,(P,T,F)*} = f^{M_0,(P,T,F)} \setminus \{t \in f^{M_0,(P,T,F)} | \mu(t) = a_i\}$[2]. □

[2]Note that we use the $\setminus$ operator here to subtract a set of elements (invisible transitions) from a sequence. This operation can be formalized in the following manner. Let σ be a sequence defined as a function $f : \mathbb{N}^+ \to D$ with D the domain, i.e. the set of elements which can be contained in the sequence. Let R be a set of elements to be removed from the sequence. Then $\sigma' = \sigma \setminus R$ is now defined as $\langle L(\sigma, R, i, 1) | i = 1, \ldots |\sigma| \rangle$ with $L(\sigma, R, i, j)$ a recursive function $(j \in \mathbb{N}^+)$:

$$\begin{aligned} L(\sigma, R, i, j) &= \emptyset \text{ if } j > |\sigma|; \\ &= L(\sigma, R, i, j+1) \text{ if } \sigma_j \in R; \\ &= L(\sigma, R, i-1, j+1) \text{ if } \sigma_j \notin R \wedge i > 1; \\ &= \sigma_j \text{ otherwise.} \end{aligned}$$

Note that it is possible that multiple firing sequences can be found to replay a trace. When a firing sequence can be found for a trace so that all transitions can be fired sequentially starting from initial marking M_0 without one of them being forced to fire, it is said that the trace fits in the Petri net. The question on how exactly a firing sequence can be constructed through a Petri net for a given trace is left open for now. Various replay strategies can be constructed in order to do so, but these will be discussed later on (in Chapter 3).

Second, there exists another common representational language to model discovered process models, namely a Causal net (C-net) [43].

Definition 1.14. Causal net. A Causal net can be expressed as a tuple $C_N = (T_C, t_s, t_e, I, O)$ where T_C is a finite set of tasks modeled by the Causal net, $I : T_C \mapsto \{X \subseteq \mathcal{P}(T_C) | X = \{\emptyset\} \vee \emptyset \notin X\}$ defines the set of possible input bindings per task (an input binding is a set of sets of activities) and $O : T_C \mapsto \{X \subseteq \mathcal{P}(T_C) | X = \{\emptyset\} \vee \emptyset \notin X\}$ defines the set of possible output bindings per task. Causal nets must have a start task $t_s \in T_C$ for which $I(t_s) = \{\emptyset\}$ and one end task $t_e \in T_C$ for which $O(t_e) = \{\emptyset\}$. □

Definition 1.15. Dependency graph. For each task $t \in T_C$, $\Box t = \bigcup(I(t))$ takes the union of all subsets in $I(t)$ and denotes the set of all input tasks, whereas $t\Box = \bigcup(O(t))$ denotes the set of output tasks of t. Based on this, a dependency graph (C_N, D) can be defined as a relation on T_C, with D the set of pairs: $\{(a, b) | a \in T_C \wedge b \in T_C \wedge (a \in \Box b \vee b \in a\Box)\}$. All tasks $t \in T_C$ in the graph (C_N, D) should lie on a path from the starting to the ending task. □

The set of sets of tasks denoting the input and output bindings (I and O respectively) are interpreted as a disjunction (between the sets of tasks) of conjunctions (between the tasks within a set). The output bindings for each task create obligations whereas input bindings resolve obligations. A "binding sequence" models an execution path through a Causal net starting and ending with the start and end task respectively and while removing all obligations created during execution. As an example, consider a task t with $I(t) = \{\{a\}, \{b\}\}$ and $O(t) = \{\{c, d\}, \{e\}\}$, meaning that this activity can only be executed when it is preceded by a or b (disjunction between sets), and that its execution creates the obligation that the task should be followed with c and d (conjunction within sets), or just e . Further details about the semantics of Causal nets can be found in [43]. Two important remarks should be mentioned, however. First, note that the semantics of Causal nets are non-local, as an output binding may create the obligation to execute an activity much later in the process. In addition, in the case of an output binding consisting of multiple sets of sets of tasks, it is not clear at the time of executing the task at hand which of the possible conjunctive "and"-sets will be resolved. As such, during execution, a state must be kept represented by multisets of pending obligations which still need to be resolved. Second, note that some descriptions and implementations of heuristic, dependency-based process discovery algorithms—such as Heuristics Miner [44]—derive models in a similar representational language, i.e. a Heuristics net, but impose an inverted interpretation on the input and output bindings, namely as a conjunction of disjunctions, meaning that $O(t) = \{\{c, d\}, \{e\}\}$ then denotes that t must be followed by e and either c or d. The intricacies of this difference will be further discussed and elaborated later on (in Chapter 5).

1.2.2 State of the Art

To close the introductory preliminaries section, we provide a concise overview of state of art regarding process discovery and conformance checking algorithms.

1.2.2.1 Process Discovery

Throughout the past decades, a plethora of process discovery technique have been proposed and developed (see Table 1.1). One of the major drivers for scholars and practitioners alike is the development of the ProM framework [45, 46], which is a support toolkit developed at TU/Eindhoven and which allows for the straightforward development of new process mining plugins. Generally speaking, process discovery algorithms can be divided along two dimensions. First, based on the approach they apply, and second, based on the structures they are able to discover and typical problems they are able to deal with. Concerning the approaches, we can identify the following items:

- *Early algorithmic approaches*, e.g. Rnet, Ktail, Markov, General DAG, B-F(k,c)-algorithm, Global partial orders, Process Miner, α-algorithm in Table 1.1.
  Most of these approaches are naive in the sense that they are not able to deal with noisy event logs, incomplete event logs, or discover structural aspects such as concurrency or loops.

- *Heuristic, dependency-based approaches*, e.g. Heuristics Miner, Fuzzy Miner in Table 1.1.
  Most of these techniques have been proposed in the form of an improvement of the substantial α-algorithm, for example to add the ability to deal with noise, enable the discovery of short loops, non-free choice constructs or duplicate tasks.

- *Genetic approaches*, e.g. Genetic Miner, Process Tree Miner in Table 1.1.
  Most of these techniques are robust to noise and able to discover a broad range of constructs without falling into "local optima", but they generally require a (very) high amount of computational time and scale badly in terms of the input event log.

- *Machine learning-based approaches*, e.g. ILP-based approach, AGNEsMiner in Table 1.1.
  These techniques apply machine learning techniques to discovery control flow.

- *Region-based approaches*, e.g. FSM Miner-Petrify, FSM Miner-Genet, Convex Polyhedra-approach in Table 1.1.
  These techniques focus on the so-called synthesis problem: constructing a process model from a description of its behavior, oftentimes represented as a Transition System or a convex hull of Parikh vectors.

For more details, we refer to [4, 47].

Table 1.1: Overview of well-known existing process discovery algorithms.

Year	Name	Author(s)
1995	Rnet, Ktail, Markov	Cook and Wolf [48, 49]
1998	General DAG	Agrawal et al. [50]
1998	B-F(k,c)-algorithm	Datta [51]
2000	Global partial orders	Manilla and Meek [52]
2002	Process Miner	Schimm [53, 54]
2003	α-algorithm	van der Aalst et al. [55]
2003	Heuristics Miner (originally: Little Thumb)	Weijters et al. [44, 56, 57]
2004	$\alpha+$-algorithm	van der Aalst et al. [58]
2004	InWoLvE	Herbst and Karagiannis [59]
2005	Multi-phase Miner	van Dongen and van der Aalst [60]
2005	Workflow Miner	Gaaloul et al. [61]
2006	DWS Mining	Greco et al. [62]
2006	Rule-based approach	Maruster et al. [63]
2006	ILP Miner	Ferreira and Ferreira [64]
2007	Genetic Miner	Alves de Medeiros et al. [65]
2007	DT Genetic Miner	Alves de Medeiros et al. [66]
2007	Fuzzy Miner	Günther and van der Aalst [67]
2007	$\alpha++$-algorithm	Wen et al. [68]
2007	DecMiner	Lamma et al. [69]
2008	AWS Mining	Greco et al. [70]
2008	Hidden Markov Model-approaches	Various authors [48, 49, 71–73]
2009	AGNEsMiner	Goedertier et al. [2]
2009	β-algorithm (or: Tsinghua α-algorithm)	Wen et al. [74]
2009	Enhanced WFMiner	Folino et al. [75]
2009	EM-approach	Ferreira and Gill [76]
2009	ILP Miner	van der Werf et al. [77]
2010	FSM Miner/Petrify	van der Aalst et al. [78, 79]
2010	FSM Miner/Genet	Carmona et al. [80]
2010	Convex Polyhedra-approach	Carmona et al. [81]
2011	Declare Miner	Maggi et al. [82, 83]
2010	Heuristics Miner++	Burattin and Sperduti [84, 85]
2010	Flexible Heuristics Miner	Weijters and Ribeiro [86]
2012	Stream-aware Heuristics Miner	Burattin and Sperduti [87]
2012	Process Log Tree Miner	van der Aalst and Buijs [88]
2012	SMT Miner	Solé and Carmona [89]
2013	Disco	Günther and Rozinat [90]
2013	MinerFul++	Di Ciccio et al. [91]
2013	UnconstrainedMiner	Westergaard et al. [92]
2014	Declarative genetic-approach	vanden Broucke et al. [93]
2014	Inductive Miner	Leemans et al. [94]
2014	Hybrid Miner	Maggi et al. [95]
2014	ProDiGen	Vázquez-Barreiros et al. [96]
2014	BPMN Miner	Conforti et al. [97]
2014	Fodina	vanden Broucke et al. (Chapter 5)
2014	Multi-Paradigm Miner	De Smedt et al. [98]

1.2.2.2 Two Simple Process Discovery Algorithms

This section provides a brief description of two simple process discovery algorithms. The first algorithm we describe is the α-algorithm [55]:

Definition 1.16. α-algorithm. Input: an event log L. Output: a Petri net (P, T, F). A footprint of the event log is established as follows:

- $\forall a, b \in A_L : [\exists \sigma \in L, i \in \{1, \dots |\sigma|\} : [\sigma_i = a \wedge \sigma_{i+1} = b]] \iff a >_L b$
- $\forall a, b \in A_L : [a >_L b \wedge b \not>_L a] \iff a \rightarrow_L b$
- $\forall a, b \in A_L : [a \not>_L b \wedge b \not>_L a] \iff a \#_L b$
- $\forall a, b \in A_L : [a >_L b \wedge b >_L a] \iff a ||_L b$

Next, we construct (P, T, F) so that $\forall a \in A_L : [t_a \in T]$, $T_I \subseteq T = \{t_a \in T | \exists \sigma \in L : [\sigma_1 = t_a]\}$, $T_O \subseteq T = \{t_a \in T | \exists \sigma \in L : [\sigma_{|\sigma|} = t_a]\}$; $X = \{(A, B) | A \subseteq T \wedge A \neq \emptyset \wedge B \neq \emptyset \wedge \forall a \in A, b \in B : [a \rightarrow_L b] \wedge \forall a, b \in A : [a \#_L b] \wedge \forall a, b \in B : [a \#_L b]\}$; $Y = \{(A, B) \in X | \forall (A', B') \in X : [A \subseteq A' \wedge B \subseteq B'] \implies (A, B) = (A', B')\}$; $\forall (A, B) \in Y : [p_{(A,B)} \in P]$,$i \in P$, $o \in P$; $F = \{(a, p_{(A,B)}) | (A, B) \in Y \wedge a \in A\} \cup \{(p_{(A,B)}, b) | (A, B) \in Y \wedge b \in B\} \cup \{(i, t_a) | t_a \in T_I\} \cup \{(t_a, o) | t_a \in T_O\}$. □

The α-algorithm can discover a large class of WF-nets under the assumption that the given event log is complete with respect to the relation $>_L$ and free of noise. Note that the α-algorithm actively tries to discover parallel behavior based on traces of atomic events. Most commercial tools, however, apply a strategy where parallel behavior is induced based on overlapping activities with duration, and all other binary sequences are regarded as non-parallel flows. Disco [90], for instance, applies a similar technique in order to derive a process model which can most easily be expressed as a C-net:

Definition 1.17. Naive C-net discovery. Input: an event log L. Output: a Causal net $C_N = (T_C, t_s, t_e, I, O)$.

- We assume all events σ_i to have start and completion time-stamps, provided by $\mathtt{StartTime}(\sigma_i)$ and $\mathtt{EndTime}(\sigma_i)$. If an event is atomic, the time of completion is equal to the start time.

- Prepend and append each trace $\sigma \in L$ with a unique start s and ending event e ($\sigma = \langle s, \sigma_1, \ldots, \sigma_{|\sigma|}, e \rangle$) with start and completion timings such that the start event comes before all events in the event log and the ending event comes after all events in the event log. Both these events are atomic.
- $\forall a \in L_A : [t_a \in T_C]$, t_s and t_e follow from the prepended and appended activities.
- We create a dependency graph D as follows: $\forall \sigma \in L, i = \{2, \ldots, |\sigma|\}$, we find $\mathtt{ArgMax}_j(j < i \wedge \mathtt{EndTime}(\sigma_j) < \mathtt{StartTime}(\sigma_i))$ and add (σ_j, σ_i) to D. Note that this will create dependency arcs between each successive activity, unless the activity overlaps which its predecessor, in which case the activity is dependent on the closest completed predecessor.
- To convert the dependency graph to a C-net, we establish input and output bindings so that. $I(t) = \mathcal{P}(\{a \in T_C | (a, t) \in D\}) \setminus \emptyset$ (the most lenient possibility regarding input combinations) and $O(t) = \{e | e \in \mathcal{P}(\{a \in T_C | (t, a) \in D\}) \setminus \emptyset \wedge \exists \sigma \in L : [\forall x, y \in e : [x \neq y \wedge x \in \sigma \wedge y \in \sigma \wedge (\mathtt{StartTime}(x) < \mathtt{EndTime}(y) < \mathtt{EndTime}(x) \vee \mathtt{StartTime}(x) < \mathtt{StartTime}(y) < \mathtt{EndTime}(x))]]]\}$, i.e. bindings are added for each combination of dependent activities which were observed in an overlapping fashion.

Note: in case events do not contain start and completion times but are instead split up in two atomic events, a matching step needs to be performed to match starting and completion events (consider for instance a trace $\langle \mathtt{start}, a_{\mathtt{start}}, a_{\mathtt{start}}, a_{\mathtt{end}}, a_{\mathtt{end}}, \mathtt{end} \rangle$). Disco, for instance, applies a FIFO approach to match the first encountered ending event with the first encountered starting event. □

1.2.2.3 Conformance Checking

Next to process discovery, another common process mining task pertains to process conformance checking, where existing process models are compared with real-life behavior as captured in event logs so as to measure how well a process model performs with respect to the actual executions of the process at hand [99]. As such, the "goodness" of a process model is typically assessed over the following four quality dimensions [47, 99]: fitness (or: recall, sensitivity), indicating the ability of the process model to correctly replay the observed behavior;

precision (or: appropriateness), i.e. the model's ability to disallow unwanted behavior; generalization, which indicates the model's ability to avoid overfitting; and finally, simplicity (or: structure, complexity, comprehensibility), stating that simpler process models should be preferred above more complex ones if they are able to fit the observed behavior just as well, thus embodying the principle of Occam's Razor. Table 1.2 provides an overview of well-known conformance checking algorithms.

1.3 Artificial Negative Events

The field of process mining is faced with some typical difficulties. For instance, process discovery algorithms have to deal with the discovery of complex structural behavior such as non-free choice, invisible activities (which were executed but not recorded in an event log) and duplicate activities, which make the hypothesis space of such algorithms harder to navigate. Secondly, process mining algorithms have to be able to work with event logs which are incomplete, or contain noisy, incorrect data. This often leads to the discovery of models which overfit the given log, a problem which is also closely related to the fact that the learning tasks in process mining are most commonly limited to the harder setting of unsupervised learning, since information about state transitions (e.g. starting, completing) that were prevented from taking place is often unavailable in real-life event logs and consequently cannot guide the learning task [2]. This latter issue will be our focus throughout the first part of this dissertation, namely the absence of negative examples in event logs, i.e. the absence of *negative events*.

Several methods have been proposed in literature to tackle the issue of negative events in process event logs, often in the context of supervised classification techniques and the application of machine learning on process discovery, the latter of which requires the presence of negative examples in order to distinguish the right hypothesis from an infinite number of possible grammars that fit the positive examples [115]. Note that event logs sometimes do contain negative events in a natural manner (*negative events in a natural manner*). Access and security logs, for example, contain detailed information about users that were refused authorization to perform a particular task, or to access a particular resource. In many cases, however, information systems do not reveal or capture these negative events, so that authors have pursued a variety of approaches to deal with this issue. Maruster et al. [63], for instance were among the first to investigate the use

Table 1.2: Overview of well-known conformance checking algorithms.

YEAR	NAME (SYMBOL)	AUTHOR(S)	MODEL INPUT TYPE	QUALITY DIMENSION
2006	Continuous Parsing Measure (CPM)	Weijters et al. [44]	Heuristic net	Recall
2006	Parsing Measure (PM)	Weijters et al.. [44]	Heuristic net	Recall
2006	Improved Continuous Semantics ($\mathsf{PF}_{\mathsf{complete}}$ or ICS)	Alves de Medeiros et al. [66]	Heuristic net	Recall
2006	Completeness	Greco et al. [62]	Workflow schema	Recall
2006	Soundness	Greco et al. [62]	Workflow schema	Precision
2008	Fitness (f)	Rozinat and van der Aalst [100]	Petri net	Recall
2008	Proper Completion (p^{PC})	Rozinat and van der Aalst [100]	Petri net	Recall
2008	Behavioral Appropriateness (a_B)	Rozinat and van der Aalst [100]	Petri net	Precision
2008	Advanced Behavioral Appropriateness ($a_B^{'}$)	Rozinat and van der Aalst [100]	Petri net	Precision
2008	Structural Appropriateness (a_S)	Rozinat and van der Aalst [100]	Petri net	Simplicity
2008	Advanced Structural Appropriateness ($a_S^{'}$)	Rozinat and van der Aalst [100]	Petri net	Simplicity
2009	Behavioral Recall (r_B^p or r_B)	Goedertier et al. [2], vanden Broucke et al. [101]	Petri net	Recall
2009	Behavioral Specificity (s_B^n)	Goedertier et al. [2]	Petri net	Precision
2010	Behavioral Profile-based Conformance metrics (MCC, CCC, etc.)	Weidlich et al. [102–104]	Petri net	Recall
2010	ETC Precision ($\mathsf{etc_p}$)	Muñoz-Gama et al. [105, 106]	Petri net	Precision
2011	Behavioral Precision (p_B)	De Weerdt et al. [107]	Petri net	Precision
2011	F-measure (F_1)	De Weerdt et al. [107]	Petri net	Recall and Precision
2011	(Average) Alignment Based Trace Fitness (f_a, f_a^{avg})	van der Aalst et al. [108–111]	Petri net	Recall
2012	Alignment Based Precision (p_A)	Adriansyah et al. [109]	Petri net	Precision
2012	Alignment Based Probabilistic Generalization (g_A)	Adriansyah et al. [109]	Petri net	Generalization
2012	One Align Precision (a_p^1)	Adriansyah et al. [112]	Petri net	Precision
2012	Best Align Precision (a_p)	Adriansyah et al. [112]	Petri net	Precision
2012	Probabilistic Generalization	Buijs et al. [113]	Process tree	Generalization
2007	Various simplicity metrics	Mendling et al. [114]	Petri net	Simplicitu
2014	Weighted Behavioral Precision (p_B^w)	vanden Broucke et al. [101]	Petri net	Precision
2014	Weighted Behavioral Generalization (g_B^w)	vanden Broucke et al. [101]	Petri net	Generalization
2014	Improved Continuous Semantics (ICS)	vanden Broucke et al. (Chapter 5)	Causal net	Fitness
2014	Flow metric (Flow)	vanden Broucke et al.(Chapter 5)	Causal net	Fitness
2014	Fuzzy metric (Fuzzy)	vanden Broucke et al.(Chapter 5)	Causal net	Fitness
2014	Recall metric (Recall)	vanden Broucke et al.(Chapter 5)	Causal net	Fitness
2014	Parsing Measure (PM)	vanden Broucke et al.(Chapter 5)	Causal net	Fitness
2014	Fitting Single Trace Measure (PM^1)	vanden Broucke et al.(Chapter 5)	Causal net	Fitness

of rule-induction techniques to predict dependency relationships between activities. The authors use the uni-relational classification learner RIPPER [116, 117] on a table of direct metrics for each process activity in relation to other activities, which is generated in a pre-processing step. This conversion is performed in order to deal with the absence of negative events, which are thus not considered in the discovery task (*ignore negative events*). Next, Ferreira and Ferreira [64] apply a combination of inductive logic programming and partial-order planning techniques to process mining. In this context, negative events are collected from users and domain experts who indicate whether a proposed execution plan is feasible or not, iteratively combining planning and learning to discover a process model (*collect negative events from domain experts*). Lamma et al. [69, 118–120] apply an extension of logic programming, SCIFF, towards declarative process discovery. Process discovery techniques using SCIFF also requires the presence of negative events (or "non-compliant" traces), but unlike the approach of Ferreira and Ferreira, the authors assume the presence of negative events, without providing an immediate answer to their origin (*assume negative events as given*). Goedertier et al. [2] represent the process discovery task as a multi-relational first-order classification learning problem and use the TILDE inductive logic programming learner for their AGNEsMiner algorithm to induce the discriminating preconditions that determine whether an event can take place or not, given a history of events of other activities. These preconditions are then converted to a graphical model after applying a pruning and post-processing step. To guide the learning process, an input event log is supplemented with induced negative events by replaying the positive events of each process instance and by checking if a state transition of interest corresponding to a candidate negative event could occur, more specifically by investigating if other traces can be found in the event log which do allow this state transition, and present a similar history of completed activities. If no such similar trace can be found, a negative event is induced (*artificially induced negative events*). This basic process of artificially generating negative events is described in a more formal way by Algorithm 1.1.

Generating a robust set of artificial negative events boils down to finding an optimal set of negative examples under the counteracting objectives of correctness and completeness. Correctness implies that the generation of false negative events has to be prevented, while completeness entails the induction of "non-trivial" negative events, that is, negative events which are based on constraints imposed by complex structural behavior, such as non-free choice constructs. The existence of the trade-off between these two goals is due to a "completeness assumption" one has to make over a given event log when generating artificial

Algorithm 1.1 Basic artificial negative event generation in an event log.

```
Input: An event log $L$
Output: A set $N$ of induced artificial negative events
 1: $N := \emptyset$
 2: for all $\sigma \in L$ do
 3:     for all $\sigma_i \in \sigma$ do
 4:         for all $a \in A_L \setminus \{\sigma_i\}$ do
 5:             if $\nexists \tau \in L : [\sigma \neq \tau \land a = \tau_i \land \forall l \in \{1, \ldots, i-1\} : [\sigma_l = \tau_l]]$ then
 6:                 % Record negative event at position $i$ in trace $\sigma$ for activity $a$:
 7:                 $N := N \cup \{(\sigma, i, a)\}$
 8:             end if
 9:         end for
10:     end for
11: end for
12: return $N$
```

negative events. Under its most strict form, the completeness assumption states that the given event log contains all possible traces that can occur. Without some assumption regarding the completeness of a given event log, it would be impossible to induce any negative events at all, since no single candidate negative event can be introduced in the knowledge that the given log does not cover all possible cases. Note that process discovery algorithms make a similar assumption, in order to derive models which are not overly general. Under the strictest completion assumption, however, the expectation is that all possible trajectories in the process model to be learned have corresponding process instances in the event log. Otherwise, a large number of negative events could be falsely induced, due to the fact that there are not enough similar traces present in the event log in order to prove the possible occurrence of a considered state transition and consequently disprove the validity of a candidate negative event. A large number of process instances is thus required in order to correctly induce negative events. (This is the problem faced by the naive approach in Algorithm 1.1.) As mentioned in the introduction, processes containing concurrent (parallelism) and recurrent (loops) behavior exponentially increase the number of allowed traces which can occur, so that the completeness assumption can become problematic. Obviously, the question on how we can make the artificial negative event induction algorithm robust to different levels of log completeness is an important challenge to address. To do so, AGNEsMiner [2] implements a window size parameter to restrict the number of events which are compared when evaluating a candidate negative event. In the work presented in this dissertation, we aim to make the completeness assumption more configurable and the generation procedure more robust so that the induction of false negative events in cases where an event log does not capture all possible behavior is prevented (correctness), while also remaining able to derive "non-trivial" negative events, that is, negative events following from complex structural behavior (completeness).

Enhancing a process log with a correct and complete set of negative events is useful for a number of reasons. First, supervised learners can now be employed in order to perform a process discovery task. Multi-relational learning techniques have already been applied in this context, using inductive logic programming in order to learn a process model from a given event log supplemented with negative events. We refer to [119–123] for a detailed overview of inductive logic programming and its applications in the field of process mining. Second, event logs supplemented with negative events can also be applied towards evaluation purposes, in order to assess fitness, precision and generalization capabilities of process discovery algorithms. For instance, the metrics in [2, 107] make use of event logs containing negative events to compose a confusion matrix in concordance with standard metric definitions in the field of data mining. In Chapter 3, we will extend this with more robust precision and generalization metrics. Third, since negative events capture which actions in a process could not occur, compliance and conformance related analysis tasks present themselves as a natural area of application for negative events. For example, auditors can cross-check a set of induced negative events with the expected behavior of real-life processes to determine if prohibited actions were indeed captured in this generated set of rejected activity state transitions. We will explore this in more detail in Chapter 4. Another specific example is access rule mining. Often, users are not so much interested in the actual control policy already present in a specific process, as this policy might already have been formally specified, but rather in a more restrictive policy that reveals which access rules are unnecessary. Negative events can then stipulate that a particular agent did never perform a particular activity at a given time, even although it could be the case that this agent officially had the rights to do so. In this way, learners could distinguish access rules that are actually needed, instead of leaving the establishment of such rules and modifications of policies to modelers and business practitioners alone. As shown hereafter, our contributions provide some important benefits with regard to correctness and completeness when compared with earlier techniques, allowing to improve the obtained results when executing the aforementioned analysis tasks.

Chapter 2

Improved Artificial Negative Event Induction

"Positive anything is better than negative nothing."
– Elbert Hubbard

2.1 Introduction

As we have discussed in the introductory chapter, the first part of this dissertation focuses on the problem of artificial negative event induction, to use in situations where only positive events are provided in a given event log.

Recall that inducing a robust set of artificial negative events boils down to finding an optimal set of negative examples under the counteracting objectives of correctness and completeness. Correctness implies that the generation of false negative events has to be prevented, while completeness entails the induction of "non-trivial" negative events, that is, negative events which are based on constraints imposed by complex structural behavior, such as non-free choice constructs. The existence of the tradeoff between these two goals is due to the completeness assumption made over an event log when generating artificial negative events. Under its most strict form, a completeness assumption requires that the given event log contains all possible traces that can occur. Without some assumption regarding the completeness of a given event log, it would be impossible to induce any negative events at all, since no single candidate negative event can be introduced in the knowledge that the given log does not cover all possible cases.

Note that process discovery algorithms make a similar assumption, in order to derive models which are not overly general. Assuming a strict completeness assumption as defined above is, however, often unrealistic in practice for the following reasons. First, when loops can occur in the process model at hand, the number of possible traces can be infinite, so that no single bounded event log can possibly contain all trace variants. Second, when parallelism is present, an exponential number of execution paths can be defined, corresponding with the order in which the concurrent task are handled. Finally, even without loops or parallelism but just N binary choices, the number of possible traces can still be 2^N. Therefore, many process discovery algorithms make a weaker completeness assumption. For example, the formal α-algorithm [55] only derives information from explicit dependencies, meaning that an event log that contains each possible sequence of activities (one directly following the other) in one of its traces suffices. Remark that this does not entail that the α-algorithm will be able to correctly mine *each* model using just this information, but only that this is the *initial assumption* made over the given input.

In this chapter, we outline an artificial negative event generation method based on a technique first introduced in the AGNEsMiner process discovery algorithm [2]. In the original version of the algorithm, a configurable completeness assumption is defined by proposing a window size parameter and a negative event injection probability. This chapter outlines a first collection of improvements in order to make this completeness assumption more configurable and the generation procedure more robust so that the introduction of falsely induced negative events in cases where an event log does not capture all possible behavior is prevented (correctness), while also remaining able to derive "non-trivial" negative events, that is, negative events following from complex structural behavior (completeness). Ensuring a correct induction of negative events proves especially helpful in practice when these events are subsequently used for evaluation purposes. Goedertier et al. [2] for instance define two evaluation metrics to assess recall and specificity (i.e. the amount of behavior present in the event log which is captured by the proposed process model) using negative events. De Weerdt et al. [107] additionally define a precision measure based on negative events to gauge whether a mined process model does not underfit the behavior present in the event log. In Chapter 3, we will also propose a novel precision and generalization measure based on negative events.

To summarize, the goal of the improvements presented hereafter is inherently related to the correctness-completeness tradeoff as described above. First, we always want to allow for a parameter configuration so that the completeness assumption made by the event generation algorithm is as weak as or weaker than the completeness assumption made by the process mining technique(s) used in a subsequent step (with the goal of either discovery or evaluation), so that *no incorrect* artificial negative events are introduced which could throw off process mining techniques using these negative examples or skew evaluation results (correctness). Second, we want to generate *all correct* negative events, especially those following from complex behavior, so that these artificial negative events provide valuable information for subsequent discovery algorithms or allow to distinguish performance differences between discovery algorithms in an evaluation context, for instance (completeness). Although these two goals counteract each other (due to the completeness assumption made), we show that the proposed improvements help to improve both objectives in a simultaneous manner.

2.2 Preliminaries

2.2.1 Related Work

Several methods have been proposed in literature to tackle the issue of negative events in process event logs, often in the context of supervised classification techniques and the application of machine learning on process discovery, the latter of which requires the presence of negative examples in order to distinguish the right hypothesis from an infinite number of possible grammars that fit the positive examples [115].

As recalled from the introductory chapter, related literature lists the following ways on how negative events can "present" themselves:

- *Negative events in a natural manner.* some event logs contain negative events in a natural manner. Access and security logs, for example, contain detailed information about users that were refused authorization to perform a particular task, or to access a particular resource.
- *Ignore negative events.* Maruster et al. [63] were among the first to investigate the use of rule-induction techniques to predict dependency relationships between activities. To do so, the uni-relational classification learner

RIPPER [116, 117] is applied on a table of direct metrics for each process activity in relation to other activities, which is generated in a pre-processing step. This conversion is performed in order to deal with the absence of negative events, which are thus not considered in the discovery task.

- *Collect negative events from domain experts.* Ferreira and Ferreira [64] apply a combination of inductive logic programming and partial-order planning techniques to process mining. In this context, negative events are collected from users and domain experts who indicate whether a proposed execution plan is feasible or not, iteratively combining planning and learning to discover a process model.
- *Assume negative events.* Lamma et al. [69, 118–120] apply an extension of logic programming, SCIFF, towards declarative process discovery. This also requires the presence of negative events (or "non-compliant" traces), but unlike the approach of Ferreira and Ferreira, the authors assume the presence of negative events, without providing an immediate answer to their origin.
- *Artificially induce negative events.* Goedertier et al. [2] represent the process discovery task as a multi-relational first-order classification learning problem. The authors use the TILDE inductive logic programming learner for their AGNEsMiner algorithm to induce the discriminating preconditions that determine whether an event can take place or not, given a history of events of other activities. These preconditions are then converted to a graphical model after applying a pruning and post-processing step. To guide the learning process, an input event log is supplemented with induced negative events by replaying the positive events of each process instance and by checking if a state transition of interest corresponding to a candidate negative event could occur, more specifically by investigating if other traces can be found in the event log which do allow this state transition, and present a similar history of completed activities. If no such similar trace can be found, a negative event is induced.

2.2.2 Artificial Negative Event Generation

Since we extend the negative event generation algorithm as introduced in AGNEsMiner [2], we repeat the original version of the algorithm below for the sake of completeness. Note that this chapter will use the definition of an event log and Petri net as given by the introductory chapter. We define an additional predicate $x.y$ as a shorthand to represent two subsequent event identifiers within a sequence σ, i.e. $x.y \iff \exists \sigma_i \in \sigma : [\sigma_i = x \wedge \sigma_{i+1} = y]$. We will use this predicate in a context of single sequence which is therefore left implicit.

A high level overview of the original algorithm can be given as follows. Negative events record that at a given position in an event sequence, a particular event cannot occur. At each position in each event trace in the log, it is examined which negative events can be recorded for this position. The algorithm works in three phases:

1. In a first step, frequent temporal constraints are mined from the event log.
2. Secondly, parallelism and locality information is derived from these frequent temporal constraints.
3. Finally, all negative events are induced for each grouped process instance. The technique stipulates that, to do so, the event log is made more compact by grouping process traces that have identical sequences into grouped process instances, so that searching for similar behavior in the event log can be performed more efficiently. This corresponds to the definition of a "distinct event log" as given in the introductory chapter.

We outline these three steps in more detail below.

2.2.2.1 Step 1: Mining Frequent Temporal Constraints

Frequent temporal constraints are constraints that hold in a sufficient number of sequences σ within an event log L. The following list of predicates express temporal constraints that either hold or not for a particular sequence $\sigma \in L$, with $a, b, c \in A_L$ activity types:

$$\begin{aligned}
\mathtt{Existence}(1, a, \sigma) &\iff \exists \sigma_i \in \sigma : [\sigma_i = a] \\
\mathtt{Absence}(2, a, \sigma) &\iff \nexists \sigma_i, \sigma_j \in \sigma : [i \neq j \wedge \sigma_i = a \wedge \sigma_j = a] \\
\mathtt{Ordering}(a, b, \sigma) &\iff \exists \sigma_i, \sigma_j \in \sigma : [i < j \wedge \sigma_i = a \wedge \sigma_j = b] \\
\mathtt{Precedence}(a, b, \sigma) &\iff \forall \sigma_j \in \sigma, \sigma_j = b : [\exists \sigma_i \in \sigma : [\sigma_i = a \wedge i < j] \\
\mathtt{Response}(a, b, \sigma) &\iff \forall \sigma_i \in \sigma, \sigma_i = a : [\exists \sigma_j \in \sigma : [\sigma_j = b \wedge i < j] \\
\mathtt{ChainPrec}(a, b, \sigma) &\iff \forall \sigma_i \in \sigma, \sigma_i = b : [\sigma_{i-1} = a] \\
\mathtt{ChainResp}(a, b, \sigma) &\iff \forall \sigma_i \in \sigma, \sigma_i = a : [\sigma_{i+1} = b] \\
\mathtt{ChainSeq}(a, b, c, \sigma) &\iff \exists \sigma \in \sigma : [\sigma_i = a \wedge \sigma_j = b \wedge \sigma_k = c]
\end{aligned}$$

A note to clarify: the $\mathtt{ChainPrec}(a, b, \sigma)$ predicate in itself expresses that all occurrences of activity type b must be preceded by a, but a can also precede activity types other then b. The $\mathtt{ChainResp}(a, b, \sigma)$ predicate in itself expresses that all occurrences of activity type a must be followed by b, but b can also follow activity types other then a.

For an event log L, a temporal constraint c is considered frequent if its support is greater than or equal to a predefined threshold. Let c_i and c_j be two temporal constraints $\in C_L$ the set of all derived temporal constraints for an event log L. The support for a temporal constraint can then be defined as: $\mathtt{Supp}(c_i) = \frac{|S|}{|L|}$ for which S is a set containing the sequences $\sigma \in L$ for which c_i holds. Temporal constraints can also be combined to form temporal association rules of the form $c_i \to c_j$. The support and confidence of an association rule are defined as:

$$\mathtt{Supp}(c_i \to c_j) = \mathtt{Supp}(c_i) \qquad \mathtt{Conf}(c_i \to c_j) = \frac{\mathtt{Supp}(c_i \wedge c_j)}{\mathtt{Supp}(c_i)}$$

Temporal association rules are considered frequent if their support and confidence are greater than or equal to a predefined threshold. Since some activities occur more frequently than others in some event logs, the detection of frequent patterns must not be sensitive to the frequency of occurrence of a particular activity type in the event log. Consider the case where the temporal constraint $\mathtt{ChainResp}(a, b, \sigma)$ has a large support, while in reality the subsequence $\langle a, b \rangle$ does not frequently occur in L. Consider for an example an event log containing a large number of traces that do *not* contain activity type a. When checking the rule $\mathtt{ChainResp}(a, b, \sigma)$ over all traces in $\sigma \in L$, this expression will hold true for all traces that do *not* contain a. Since these traces make up for a large part of the event log, the support of $\mathtt{ChainResp}(a, b)$ would thus be high. To detect frequent patterns in an event log, it is therefore more important to look at the confidence of the association rule $\mathtt{Existence}(1, a, \sigma) \rightarrow \mathtt{ChainResp}(a, b, \sigma)$. The following frequent temporal association rules are derived from an event log L (left implicit)[1]:

$$\begin{aligned}
\mathtt{Absence}(2,a) &\Longleftarrow \forall a \in A_L : [\mathtt{Supp}(\mathtt{Absence}(2,a,\sigma)) \geqslant t_{absence}] \\
\mathtt{Ordering}(a,b) &\Longleftarrow \forall a,b \in A_L : [\mathtt{Supp}(\mathtt{Ordering}(a,b,\sigma),L) \geqslant t_{ordering}] \\
\mathtt{Precedence}(a,b) &\Longleftarrow \forall a,b \in A_L : [\mathtt{Conf}(\mathtt{Existence}(1,b,\sigma) \\
&\qquad \rightarrow \mathtt{Precedence}(a,b,\sigma),L) \geqslant t_{succession}] \\
\mathtt{Response}(a,b) &\Longleftarrow \forall a,b \in A_L : [\mathtt{Conf}(\mathtt{Existence}(1,a,\sigma) \\
&\qquad \rightarrow \mathtt{Response}(a,b,\sigma),L) \geqslant t_{succession}] \\
\mathtt{ChainPrec}(a,b) &\Longleftarrow \forall a,b \in A_L : [\mathtt{Conf}(\mathtt{Existence}(1,b,\sigma) \\
&\qquad \rightarrow \mathtt{ChainPrec}(a,b,\sigma),L) \geqslant t_{chain}] \\
\mathtt{ChainResp}(a,b) &\Longleftarrow \forall a,b \in A_L : [\mathtt{Conf}(\mathtt{Existence}(1,a,\sigma) \\
&\qquad \rightarrow \mathtt{ChainResp}(a,b,\sigma),L) \geqslant t_{chain}] \\
\mathtt{ChainSeq}(a,b,c) &\Longleftarrow \forall a,b,c \in A_L : [\mathtt{Supp}(\mathtt{ChainSeq}(a,b,c,\sigma),L) \geqslant t_{triple}]
\end{aligned}$$

2.2.2.2 Step 2: Deriving Structural Information

Now that a set of temporal frequent constraints is constructed, it becomes possible to derive information about parallelism and locality of pairs of activities. Intuitively, parallelism between two activities $a, b \in A_L$ can be assumed when it is the case in the event log L that a frequently follows b and b frequently follows a. However, in the case of duplicate activities, a triple chain subsequence $\langle a, b, a\rangle$ or $\langle b, a, b\rangle$ could occur in serial, without the presence of parallelism. Therefore, we require these triple chain sequences not to occur, unless it can be detected that a and b are actually part of two separate, parallel length-one loops, see Figure 2.1. This is the case when subsequences like $\langle a, a, b\rangle$, $\langle b, a, a\rangle$, $\langle b, b, a\rangle$ or $\langle a, b, b\rangle$ are also frequent. $\mathtt{Parallel}(a, b)$ can consequently be defined as follows[2]:

$$\begin{aligned}
\forall a,b \in A_L :&[(\mathtt{ChainPrec}(a,b) \vee \mathtt{ChainResp}(a,b)) \wedge \\
&(\mathtt{ChainPrec}(b,a) \vee \mathtt{ChainResp}(b,a)) \wedge \\
&(\sim \mathtt{ChainSeq}(a,b,a) \wedge \sim \mathtt{ChainSeq}(b,a,b) \vee \\
&\quad \mathtt{ParallelLoop}(a,b))] \implies \mathtt{Parallel}(a,b)
\end{aligned}$$

$$\forall a,b \in A_L :[\mathtt{Parallel}(a,b)] \implies \mathtt{Parallel}(b,a)$$

$$\begin{aligned}
\forall a,b \in A_L :&[\mathtt{ChainSeq}(a,a,b) \vee \mathtt{ChainSeq}(b,a,a) \vee \mathtt{ChainSeq}(a,b,b) \vee \\
&\mathtt{ChainSeq}(b,b,a)] \implies \mathtt{ParallelLoop}(a,b)
\end{aligned}$$

[1] In the original definition, the predicates $\mathtt{Ordering}(a, b)$ and $\mathtt{ChainSeq}(a, b, c)$ do not account for the mutual distribution of the activity types a, b and c. They remain therefore sensitive to their frequency of occurrence.

[2] Remark that we use negation-as-failure ($\sim$) rather than normal logical negation ($\neg$) to denote that the absence of a frequent temporal constraint is derived from the absence of sequences in the event log that portray this behavior. In this context, the reader may ignore the exact semantic differences between the two notations, as $\sim p \implies \neg p$ under a closed-world assumption.

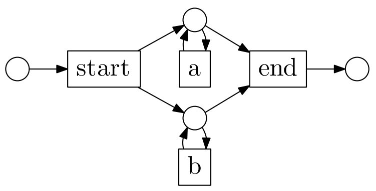


Figure 2.1: Two parallel length one loops make the induction of parallelism challenging.

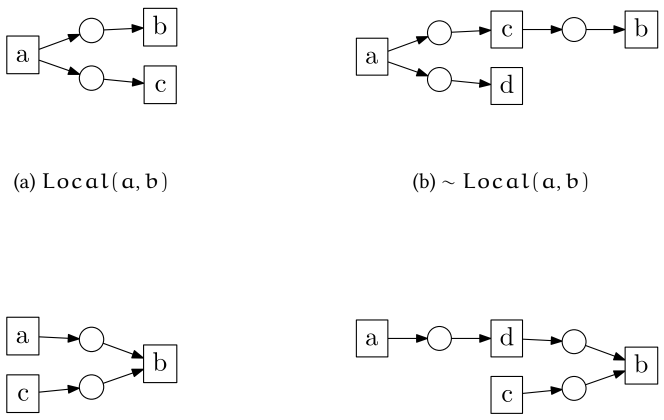
Locality (serial occurrence denoting an explicit dependency) between activities can also be derived from the frequent temporal constraints. As a rule of thumb, $\mathtt{a}$ is local to $\mathtt{b}$ if $\mathtt{a}$ is frequently followed by $\mathtt{b}$ or $\mathtt{b}$ is frequently preceded by $\mathtt{a}$:

$$\begin{aligned}
&\forall a, b \in A_L : [\mathrm{ChainPrec}(a,b) \wedge \sim \mathrm{Parallel}(a,b)] \implies \mathrm{Local}(a,b)\\
&\forall a, b \in A_L : [\mathrm{ChainResp}(a,b) \wedge \sim \mathrm{Parallel}(a,b)] \implies \mathrm{Local}(a,b)
\end{aligned}$$

Unlike parallelism, locality is not symmetric, i.e. $\mathrm{Local}(a,b)$ does not imply $\mathrm{Local}(b,a)$. In the presence of parallelism, it is highly likely that not all local relationships can be detected. Consider the situation in Figure 2.2(a). Although $\mathtt{a}$ is local to $\mathtt{b}$ it is possible that the subsequence $\langle a, b \rangle$ occurs infrequently in the event log, due to the parallelism between $\mathtt{b}$ and $\mathtt{c}$ causing sequence $\langle a, c, b \rangle$ to occur far more than $\langle a, b, c \rangle$. In such cases, however, $\mathrm{Local}(a,b)$ can be derived from the parallelism of $\mathtt{b}$ and $\mathtt{c}$. Figure 2.2(b) on the other hand depicts a situation in which such an inference cannot be made. These concerns are translated in the following derivation rule:

$$\begin{aligned}
\forall a, b \in A_L : [\exists c \in A_L : [&\mathrm{Parallel}(b,c) \wedge \sim \mathrm{Parallel}(a,c) \wedge \\
&\sim \mathrm{Parallel}(a,b) \wedge (\mathrm{ChainPrec}(a,c) \vee \mathrm{ChainResp}(a,c))] \wedge \\
&\nexists d \in A_L : [(\mathrm{ChainResp}(d,b) \vee \mathrm{ChainPrec}(d,b)) \wedge \\
&\quad \sim \mathrm{Parallel}(d,b)]] \implies \mathrm{Local}(a,b)
\end{aligned}$$

The same reasoning for an AND-split can be applied to an AND-join, as displayed in Figure 2.2(c, d), leading to the following second derivation rule:



(a) $\mathsf{Local}(a,b)$

(b) $\sim \mathsf{Local}(a,b)$

(c) $\mathsf{Local}(a,b)$

(d) $\sim \mathsf{Local}(a,b)$

Figure 2.2: Deriving locality from parallelism.

$$\begin{aligned}\forall a,b \in A :&[\exists c \in A_L : [\mathsf{Parallel}(a,c) \wedge \sim \mathsf{Parallel}(b,c) \wedge \\ &\sim \mathsf{Parallel}(a,b) \wedge (\mathsf{ChainPrec}(c,b) \vee \mathsf{ChainResp}(c,b))] \wedge \\ &\nexists d \in A_L : [(\mathsf{ChainResp}(a,d) \vee \mathsf{ChainPrec}(a,d)) \wedge \\ &\sim \mathsf{Parallel}(a,d)]] \implies \mathsf{Local}(a,b)\end{aligned}$$

2.2.2.3 Step 3: Generating Artificial Negative Events

Once parallelism and locality information is derived, negative examples can be introduced in grouped process instances by checking at any given positive event whether any other activity type in the event log could occur as an event at this position. To address the problematic nature of the completeness assumption of an event log under recurrent behavior (loops), a window size parameter ws is introduced to limit the number of events which are compared when evaluating a candidate negative event. The larger the window size, the less probable that a similar subsequence is contained by the other sequences in the event log, and the more probable that a candidate negative event will be introduced. Reducing the window size makes the completeness assumption less strict. An unlimited

Given:

σ	$\langle a, b, c, d, e, f\rangle$

$\mathsf{Parallel}(b, d)$
$\mathsf{Parallel}(c, d)$
$\mathsf{Parallel}(b, e)$
$\mathsf{Parallel}(c, e)$

Parallel variants:

$\sigma^{\|\|}$	$\langle a, b, c, d, e, f\rangle$
$\sigma^{\|\|}$	$\langle a, b, d, c, e, f\rangle$
$\sigma^{\|\|}$	$\langle a, b, d, e, c, f\rangle$
$\sigma^{\|\|}$	$\langle a, d, b, e, c, f\rangle$
$\sigma^{\|\|}$	$\langle a, d, e, b, c, f\rangle$

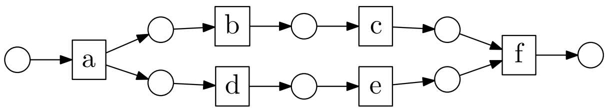


Figure 2.3: Calculating parallel variants for a trace based on derived structural information.

window size (a maximum-length comparison between sequences) results in the most strict completeness assumption.

Concurrent behavior (parallelism) can pose problems with regards to the completeness assumption of an event log as well. This issue is addressed by exploiting the previously mined parallelism information to generate parallel variants of a subsequence at hand which is used to disprove a particular candidate negative event. Taking the parallelism information into account results in a higher number of subsequences which will be used when evaluating a negative event and thus results in a weaker completeness assumption. The calculation of parallel variants is illustrated in Figure 2.3.

Negative events record that at a particular position in an event sequence, a particular event cannot occur. At each position i in each event sequence $\sigma \in L$, it is examined which negative events can be introduced at this position. Algorithm 2.1 formalizes this process. The algorithm loops over all traces $\sigma \in \bigcup L$ (the distinct event log). For each "positive trace" under consideration, we iterate over all activities $\sigma_i \in \sigma$ and check whether another event of interest (a "candidate negative event") $n \in A_L \setminus \{\sigma_i\}$ also could occur. As such, for each positive event $\sigma_i \in \sigma$, it is tested whether there exists a sequence $\tau^{||} \in \{\tau \in PV(\lambda) | \lambda \in \bigcup L \setminus \{\sigma\}\}$ in the event log in which at that point an event $\tau_i^{||} = n$ has taken place, with $PV : \sigma \in L \mapsto PV_\sigma$ a function which returns the set PV_σ containing all parallel variants of a trace and contains a similar history up until the occurrence of $\tau_i^{||}$ as the trace σ exhibits up until σ_i. If such a similar "disproving sequence" can not be found, such behavior is not present in the event log L, meaning that the candidate negative event n cannot be disproved and is added at position i in the positive sequence σ. Finally, the induced artificial negative events in the grouped traces can, naturally, be used to induce negative events in the complete event log L. Usually, a very large number of negative events can be generated, so that a probability π is introduced as a threshold for injecting negative events into the ungrouped sequences.

Algorithm 2.1 Artificial negative event generation algorithm—original AGNEsMiner implementation [2].

```
Input: An event log $L$
Input: Window size parameter $ws$
Output: A set $N$ of induced artificial negative events
1:  $N := \emptyset$
2:  for all $\sigma \in \bigcup L$ do
3:      for all $\sigma_i \in \sigma$ do
4:          for all $n \in A_L \setminus \sigma_i$ do
5:              if $\nexists \tau^{||} \in \{\tau \in PV(\lambda) | \lambda \in \bigcup L \setminus \{\sigma\}\}$ : then
6:                  $[n = \tau^{||}_i \wedge \forall l \in \{i - ws, \ldots, i-1\} : [\sigma_l = \tau^{||}_l]]$ then
7:                  % Record negative event at position $i$ in trace $\sigma$ for activity $n$:
8:                  $N := N \cup \{(\sigma, i, n)\}$
9:              end if
10:         end for
11:     end for
12: end for
13: % The negative events in $N$ are inserted in $L$ according to probability $\pi$
14: return $N$
```

Figure 2.4 illustrates how in an event log of two traces σ and τ artificial events can be generated. Note that history dependent processes generally will require a larger window size to correctly detect all non-local dependencies. In Figure 2.4 for example, an unlimited window size is used. Should the window size be limited to 1, it would no longer be possible to take into account the non-local dependency between activity pairs b−f and c−g. This underlines the tradeoff as discussed before in the introduction. Using a *higher window size leads to the generation of more valuable negative events* (either for discovery and evaluation purposes), that is, negative events derived from non-local dependencies. On the other hand, using an increased window *also leads to a more strict completeness assumption, so that candidate negative events are less likely to be disproved, leading to the possible introduction of incorrect negative examples.* The improvements presented hereafter aim to tackle this tradeoff, by making the completeness assumption more configurable and by introducing a number of techniques which allow to generate a set of negative events which is more likely to be both correct and complete.

Given:

σ	$\langle a, b, d, e, f, h\rangle$
τ	$\langle a, c, e, d, g, h\rangle$

$\mathit{Parallel}(d, e)$

Parallel variants:

$\sigma^{\|\|}$	$\langle a, b, d, e, f, h\rangle$
$\sigma^{\|\|}$	$\langle a, b, e, d, f, h\rangle$
$\tau^{\|\|}$	$\langle a, c, e, d, g, h\rangle$
$\tau^{\|\|}$	$\langle a, c, d, e, g, h\rangle$

Negative events:

σ	a	b	d	e	f	h
	b	a	a	a	a	a
	d	d	b	b	b	b
	e	e	e^{**}	d	d	d
	f	f	f	f	e	e
	h	h	h	h	h	f
	c	g	c	c	c	c
	g		g	g	g^{*}	g

τ	a	c	e	d	g	h
	b	a	a	a	a	a
	d	d	b	b	b	b
	e	e	d^{**}	e	d	d
	f	f	f	f	e	e
	h	h	h	h	f^{*}	f
	c	g	c	c	h	c
	g		g	g	c	g

*: Negative events based on non-local dependencies.
**: Incorrectly generated negative events.

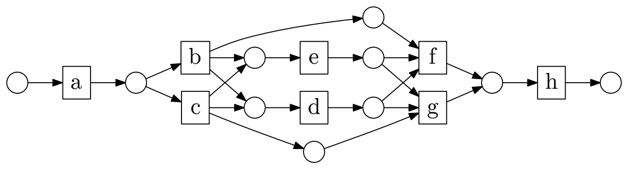


Figure 2.4: Generating artificial negative events for an event log with two traces.

2.3 Improvements

In this section, we present a number of improvements and modifications in order to create an artificial negative event generator to enhance process logs with negative event examples. We aim to make the completeness assumption more configurable and the generation procedure more robust so that the introduction of falsely induced negative events in cases where an event log does not capture all possible behavior is prevented (correctness), while also remaining able to derive "non-trivial" negative events, that is, negative events following from complex structural behavior (completeness).

We describe four extensions in this section. First, we start by introducing some new frequent temporal and structural constraints which will be used by subsequent extensions, and we modify the Ordering and ChainSeq temporal association rules so that the detection of these patterns is no longer sensitive to the frequency of occurrence of a particular activity type in the event log. Second, we show that the original window size parameter does not suffice to counter complexity caused by recurrent behavior (loops) and thus extend the variant calculation technique to also include variants based on loops which are discovered in the event log. Since processes can combine concurrent and recurrent behavior in a complex manner, it is possible that not all variants can be calculated, or that generating all variants becomes computationally infeasible, due to the recursive and exponential nature of the problem. Therefore, the third improvement introduces a dynamic window, to make the disprove procedure for a candidate negative event more strict. Using this dynamic window with a window size of 1 weakens the completeness assumption to that of the α-algorithm—meaning that an event log which contains every possible binary sequences of activities somewhere in its traces suffices to generate a fully correct set of negative events, even when no parallel or looped variants are calculated. Finally, we introduce a technique to generate negative events without performing a similarity check between traces in the event log, by relying only on discovered activity type dependency information.

2.3.1 Improvement 1: Revised Temporal Constraint Confidence Measures

We redefine the Ordering and ChainSeq temporal association rules so that the detection of these patterns is no longer sensitive to the frequency of occurrence of particular activity types. The temporal constraints describing $\mathtt{Ordering}$ and $\mathtt{ChainSeq}$ are redefined as follows:

$$\begin{aligned}
\mathtt{Ordering}(a,b,\sigma) &\iff \forall \sigma_i, \sigma_j \in \sigma, \sigma_i = a, \sigma_j = b : [j = i+1] \\
\mathtt{ChainSeq}(a,b,c,\sigma) &\iff (\forall \sigma_i \in \sigma, \sigma_i = c : [\sigma_{i-1} = b \wedge \sigma_{i-2} = a])\vee \\
&\quad\;\; (\forall \sigma_i \in \sigma, \sigma_i = b : [\sigma_{i-1} = a \wedge \sigma_{i+1} = c])\vee \\
&\quad\;\; (\forall \sigma_i \in \sigma, \sigma_i = a : [\sigma_{i+1} = b \wedge \sigma_{i+2} = c])
\end{aligned}$$

The corresponding association rules are now:

$$\begin{aligned}
\mathtt{Ordering}(a,b) &\Longleftarrow \forall a,b \in A_L : [\mathtt{Conf}(\mathtt{Existence}(1,a,\sigma)\wedge \\
&\quad\;\; \mathtt{Existence}(1,b,\sigma) \\
&\quad\quad \rightarrow \mathtt{Ordering}(a,b,\sigma), L) \geqslant t_{succession}] \\
\mathtt{ChainSeq}(a,b,c) &\Longleftarrow \forall a,b,c \in A_L : [\quad \mathtt{Conf}(\mathtt{Existence}(1,a,\sigma)\vee \\
&\quad\;\; \mathtt{Existence}(1,b,\sigma) \vee \mathtt{Existence}(1,c,\sigma)) \\
&\quad\quad \rightarrow \mathtt{ChainSeq}(a,b,c,\sigma), L) \geqslant t_{triple}]
\end{aligned}$$

We also define the following additional temporal constraints:

$$\begin{aligned}
\mathtt{Iteration}(n,\upsilon,\sigma) &\iff \upsilon \subseteq \sigma \wedge \exists \sigma_t \in \sigma : [\\
&\forall i \in \{1,\ldots,|\upsilon|\}, j \in \{1,\ldots,n\} : [\sigma_{t+i+j|\upsilon|} = \upsilon_i]]
\end{aligned}$$

$$\begin{aligned}
\mathtt{StartActivity}(a,\sigma) &\iff \sigma_i = a \\
\mathtt{EndActivity}(a,\sigma) &\iff \sigma_{|\sigma|} = a
\end{aligned}$$

With the corresponding association rules:

$$\begin{aligned}
\mathtt{Iteration}(n,\upsilon) &\Longleftarrow \forall \upsilon \subseteq \sigma, \sigma \in L, n = 2, 0 < |\upsilon| \leqslant 3 : [\\
&\quad\;\; \mathtt{Conf}(\forall \upsilon_i \in \upsilon : \mathtt{Existence}(1,\upsilon,\sigma) \\
&\quad\quad \rightarrow \mathtt{Iteration}(n,\upsilon,\sigma)) \geqslant t_{iteration}]
\end{aligned}$$

$$\begin{aligned}
\mathtt{StartActivity}(a) &\Longleftarrow \forall a \in A_L : [\mathtt{Conf}(\mathtt{Existence}(1,a,\sigma) \\
&\quad\quad \rightarrow \mathtt{StartActivity}(a,\sigma)) \geqslant t_{position}] \\
\mathtt{EndActivity}(a) &\Longleftarrow \forall a \in A_L : [\mathtt{Conf}(\mathtt{Existence}(1,a,\sigma) \\
&\quad\quad \rightarrow \mathtt{EndActivity}(a,\sigma)) \geqslant t_{position}]
\end{aligned}$$

The $\mathtt{Iteration}$ temporal constraint holds in a sequence σ if a subsequence υ iterates n times in σ. In order to keep to number of computations under control, we limit the discovery of iterations to length 3 and less, which repeat for minimally 2 times; $n = 2$ and $0 < |\upsilon| \leqslant 3$.

2.3.2 Improvement 2: Variant Calculation with Loop Discovery

The $\mathtt{Iteration}$ temporal constraint as defined in the previous section allows us to formulate a loop discovery heuristic. Loops, however, cannot immediately be derived from the set of $\mathtt{Iteration}$ temporal constraints. To see why this is the case, consider the sequence $\langle a, b, c, d, b, c, d, b, c, d, e \rangle$. The following $\mathtt{Iteration}$ constraints hold in this sequence: $\mathtt{Iteration}(2, \langle b, c, d \rangle)$, $\mathtt{Iteration}(2, \langle c, d, b \rangle)$ and $\mathtt{Iteration}(2, \langle d, b, c \rangle)$. Since we are only concerned with the loop following from the first $\mathtt{Iteration}$ constraint, with the start activity in the correct, first position, we define $\mathtt{Loop}(\upsilon)$ as such:

$$\mathtt{Loop}(\upsilon) \Longleftarrow \forall \upsilon \subseteq \sigma, \sigma \in L : [\mathtt{Iteration}(2, \upsilon) \wedge \exists a \in A_L : [\mathtt{Local}(a, \upsilon_1)]]$$

Based on the $\mathtt{Loop}$ constraints, the variant calculation method can be extended to take recurrent behavior into account as well, next to concurrent behavior. Figure 2.5 depicts an example of the generation of loop variants. The variant construction method works as follows: for each activity in a trace σ, it is investigated whether it is possible to insert additional or new loop iterations at this position (by looking at discovered $\mathtt{Local}$ and $\mathtt{Loop}$ constraints), and whether already present loop iterations can be removed. Since the addition of loop iterations could potentially lead to infinitely long sequences, some additional parameters have to be defined to guide to construction of loop variants. Firstly, in the case of inserting additional loop iterations, a parameter $\mathtt{loopAddUntil}$ defines the upper limit for the number of iterations in each loop. Setting this parameter to unlimited does not lead to generation of infinitely long sequences, but only adds enough iterations of each loop to each sequence so that for every candidate negative event considered, a loop variant trace can always be found where the window to be checked always contains loop iterations which could occur up until that point. Furthermore, a boolean parameter $\mathtt{loopAddFromZero}$ governs if loop iterations are to be added at insert positions where no iterations of the considered loop are yet present. Secondly, for the removal of iterations, a parameter $\mathtt{loopRemoveUntil}$ defines the lower limit for the number of iterations in each loop. Setting this parameter to zero completely removes loops where possible. A large number of variants can often be calculated for given sequences in an event

Given:

σ	$\langle a, c, f \rangle$

$\mathtt{Local}(a, b)$
$\mathtt{Local}(c, d)$
$\mathtt{Loop}(b)$
$\mathtt{Loop}(d, e)$
$\mathtt{MinLoopOccurrence}(0, b)$
$\mathtt{MinLoopOccurrence}(0, d, e)$
$\mathtt{MaxLoopOccurrence}(3, b)$
$\mathtt{MaxLoopOccurrence}(2, d, e)$

Loop variants:

σ^o	$\langle a, c, f \rangle$
σ^o	$\langle a, b, c, f \rangle$
σ^o	$\langle a, b, b, c, f \rangle$
σ^o	$\langle a, b, b, b, c, f \rangle$
σ^o	$\langle a, c, d, e, f \rangle$
σ^o	$\langle a, b, c, d, e, f \rangle$
σ^o	$\langle a, b, b, c, d, e, f \rangle$
σ^o	$\langle a, b, b, b, c, d, e, f \rangle$
σ^o	$\langle a, c, d, e, d, e, f \rangle$
σ^o	$\langle a, b, c, d, e, d, e, f \rangle$
σ^o	$\langle a, b, b, c, d, e, d, e, f \rangle$
σ^o	$\langle a, b, b, b, c, d, e, d, e, f \rangle$

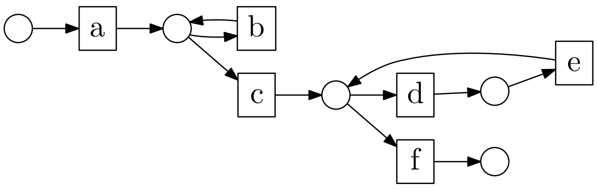


Figure 2.5: Calculating loop variants for a trace based on derived structural information.

log, leading to a higher number of traces which will be used when evaluating a negative event, thus resulting in less incorrect cases.

Note that, instead of having to specify a value for $\mathtt{loopAddUntil}$ and $\mathtt{loopRemoveUntil}$ manually, it is also possible to set these values based on the minimum and maximum number of subsequent iteration occurrences within a sequence for each discovered loop. As such, for each discovered $\mathtt{Loop}(\upsilon)$ constraint, we can also define corresponding $\mathtt{MinLoopOccurrence}(i, \upsilon)$ and $\mathtt{MaxLoopOccurrence}(j, \upsilon)$ constraints. The advantage of using this information for the calculation of loop variants is that each distinct loop can now be grown or shrunk according to the real behavior present in the log (instead of using global parameters over all loops), but note that this makes the completeness assumption more strict in the sense that the minimum and maximum possible iterations of each loop should indeed be present in the given event log.

Figure 2.6 depicts an example in order to demonstrate the usefulness of loop variant calculation when generating negative events. Although the window size parameter has been set to 1 for the generation of negative events in order to handle recurrent behavior, incorrect cases are still introduced in the event log due to the strict position comparison between the window of the positive trace at hand and the window of the trace used to disprove the negative event currently under consideration, denoted by $n = \tau_i^{||}$ in Algorithm 2.1. It is this strict comparison which causes the interesting dissimilarity in the set of negative events for σ and τ in Figure 2.6: whereas σ does not include b as a negative event in the previous-to-last position (at the position of activity c), τ does falsely introduce b as a negative example at the position of activity d, due to the difference in lengths of the two sequences caused by the loop. Furthermore, the original algorithm is not able to discover the recurrent behavior in the underlying process, leading to the rejection of completion of c and d at each position where b occurs. Using the loop variant calculation explained above avoids the generation of these incorrect cases. Moreover, it is no longer required to use the lowest possible window (size 1), so that non-local, history-dependent behavior—if present—can again be captured by the generated negative events.

Finally, note that the generation of incorrect examples is yet not completely avoided when using loop variant calculation in the example of Figure 2.6. σ, for example, still rejects the completion of c activities after the completion of a. This is not due to a particular intricacies of loop calculation (indeed, similar behavior can be seen in Figure 2.4) for the parallel variants, but rather to the following condition in Algorithm 2.1: $\lambda \in \bigcup L \setminus \{\sigma\}$. This condition prohibits variants of the current positive trace under consideration (σ) to be used as a disproving trace for a particular candidate negative event. As such, we are able to introduce a boolean parameter $useOwnTrace$, leading to the removal of all incorrect negative events.

Given:

σ	⟨a, b, b, c, e⟩
τ	⟨a, b, b, b, b, d, e⟩

Loop(b)

Negative events:

σ	a	b	b	c	e		
	b	a	a	a	a		
	c	c*	c*	d*†	b		
	d	d*†	d*†	e	c		
	e	e	e		d		
τ	a	b	b	b	b	d	e
	b	a	a	a	a	a	a
	c	c*†	c*†	c*†	c*†	b*†	b
	d	d*	d*	d*	d*	c*†	c
	e	e	e	e	e	e	d

*: Incorrectly induced negative events without using loop variant calculation, with window size = 1.
†: Incorrect negative events that are no longer present when using loop variant calculation.

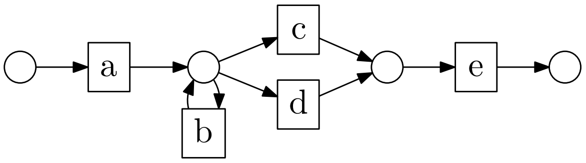


Figure 2.6: Generating artificial events for an event log containing recurrent behavior. Without the loop variant calculation extension, incorrect negative events are introduced in the event log, even although the window size parameter is set to 1.

2.3.3 Improvement 3: Dynamic Windows

Even when both parallel and loop variants are considered, incorrect negative events could still be induced, even when using the smallest possible window size of length 1. The following reasons can explain this statement. Firstly, when generating loop variants, we have only considered loops with a maximum length of 3. Although it is possible to adjust this parameter so that longer loops can be discovered, the number of checks which have to be performed rises dramatically when increasing this parameter. Secondly, even when longer loops are looked for, potential loops might still remain undiscovered due to the possible recursive nature of parallel and loop constructs. Consider for example the loop ⟨a, B, d⟩, with B a subsequence containing a parallel construct composed of activities b and c, leading to the following possible complete sequence: ⟨start, a, b, c, d, a,

$c, b, d, a, b, c, d, a, c, b, d, end\rangle$. Even in the case when we adjust the discovery of temporal constraints to mine $\texttt{Iteration}$ constraints with length 4, the loop in the sequence above will never be discovered, since the concurrency construct causes neither the $\langle a, b, c, d\rangle$ or $\langle a, c, b, d\rangle$ subsequences to repeat two times in a sequential manner. The same problem exists when loops are nested. Thirdly, even when all constructs could be detected—or given—the generation of variants still remains an exponentially complex problem, meaning that a rise in a number of concurrency or recurrence constructs more than proportionally increases the number of variants which can be generated, especially when considering the possibility that parallel variants could be calculated for previously calculated loop variants, loop variants for previously calculated parallel variants, and so on, making the variant generation method a recursive problem as well.

Instead of trying to deal with the above problems by adjusting the $\texttt{Loop}$ and $\texttt{Iteration}$ constraints to take into account concurrency, we limit the variant calculation to the simple heuristics as described above, with a maximum loop detection length of 3, and a variant calculation scheme where the parallel variants are calculated first and loop variants are generated afterward over this set; $AV : \sigma \in L \mapsto \{\tau \in LV(\kappa) | \kappa \in PV(\sigma)\}$ with $PV : \sigma \in L \mapsto PV_\sigma$ a function which returns the set PV_σ containing all parallel variants of a trace, and with $LV : \sigma \in L \mapsto LV_\sigma$ a function which returns the set LV_σ containing all recurrent variants of a trace. To deal with the problem of complex constructs leading to possible incorrect negative events, we adjust the window based comparison algorithm to remove the strict position requirement: $n = \tau_i^{||}$ in Algorithm 2.1. Instead, we compare the window of the original "positive" trace with each possible window in the "candidate disproving" trace, now denoted as $\tau^{\sim}$, meaning each sequence of events before an event with activity type equal to the activity type of the current candidate negative event under consideration. An example can clarify this principle. Consider the positive trace $\sigma = \langle a, b, c, d, f\rangle$ and $\tau^{\sim} = \langle a, b, c, b, c, e, f\rangle$ a candidate disproving trace under consideration. Assume we are currently checking to see if candidate activity e could also occur instead of d (i.e. σ_4) in σ. In cases where a strict window is used, with size equal to, say, 2, the candidate disproving trace $\tau^{\sim}$ fails to disprove the negative event, since $t_4^{\sim} \neq e$. Instead of doing so, we now compare the window $\langle b, c\rangle$ in σ with each possible window in $\tau^{\sim}$. In this case, the window of size 2 before activity $\tau_6^{\sim} = e$ in $\tau^{\sim}$ is also equal to $\langle b, c\rangle$. This leads to a correct rejection of the candidate negative event. The complete modified algorithm using a dynamic window is listed in Algorithm 2.2.

Note that, depending on the window size configured, cases might now exist where the size of the window in the original trace σ is unequal to the size of

Algorithm 2.2 Artificial negative event generation algorithm—with dynamic window approach.

```
Input: An event log $L$
Input: Window size parameter $ws$ and minimum windows size parameter $mws$
Input: Boolean parameter $useOwnTrace$
Input: Configuration for variant generating function $AV$ (generate loops yes/no, generate parallel variants
    yes/no, max loops, min loops, add loops from zero length)
Output: A set $N$ of induced artificial negative events
 1: $N := \emptyset$
 2: for all $\sigma \in \bigcup L$ do
 3:     for all $\sigma_i \in \sigma$ do
 4:         for all $n \in A_L \setminus \sigma_i$ do
 5:             $L' := \bigcup L$
 6:             if $useOwnTrace$ then
 7:                 $L' := L' \setminus \{\sigma\}$
 8:             end if
 9:             if $\nexists \tau \in \{\tau \in AV(\lambda) | \lambda \in L' : [\exists \tau_j \in \tau : [\tau_j = n \wedge ((minws >= j-1) \vee$
    $(minws = -1 \wedge j >= i)) \wedge \forall l \in \{1, \ldots, ws\} : [\sigma_{i-l} = \tau_{j-l}]]]$ then
10:                 % Record negative event at position $i$ in trace $\sigma$ for activity $n$:
11:                 $N := N \cup \{(\sigma, i, n)\}$
12:             end if
13:         end for
14:     end for
15: end for
16: % The negative events in $N$ are inserted in $L$ according to probability $\pi$
17: return $N$
```

a window in a candidate disproving trace $\tau^{\sim}$. In cases where the window in $\tau^{\sim}$ is larger than the window in σ, an effective window with size equal to the window used in σ is used. When the window in $\tau^{\sim}$ is smaller than the window in σ, using the smallest window could potentially lead to the rejection of negative events, even when a high window size parameter was set. An example can help to illustrate this. Consider again the positive trace $\sigma = \langle a, b, c, d, f \rangle$ and $\tau^{\sim} = \langle a, b, c, b, c, e, f \rangle$ the candidate disproving trace under consideration. Assume now we are currently checking to see if candidate activity b could also occur instead of d in σ. Based on this information, two windows can be defined in $\tau^{\sim}$ which can serve to check the validity of the candidate negative event at hand: $\langle a \rangle$ and $\langle a, b, c \rangle$. For the latter, no problem exists, as this window is as long as the window $\langle a, b, c \rangle$ in σ. On the other hand, the other window ($\langle a \rangle$) is smaller than the window in σ, and thus the similarity check between these two windows might differ from the actual window size parameter which was configured by the user. Although this can never lead to the generation of additional incorrect negative events, it can lead to skipped generation of additional correct negative events, so that we define a parameter $minws$ to denote a minimum required length for a window to be used in the negative event rejection procedure. This parameter can be configured so that the window in $\tau^{\sim}$ should be at least the same size as the current window in σ.

Using a dynamic window now allows us to dramatically weaken the completeness assumption made by the artificial negative event generation process. Using a dynamic window with size 1 indeed assumes only that each binary sequence of two activities is somewhere present in the given event log, or can be generated from the given event log using parallel and loop variant calculation as described above. This weakens the completeness assumption equal to the one made by the formal α-miner learner.

Finally, remark the absence of a "forward window"; we only consider the history of completed events when investigating which activities could not be completed at a certain point in the process instance. The reason for this is straightforward: at the time of investigating a current state transition, the future of the process instance at hand is still unknown. Therefore, only historical facts can be considered in the negative event generation process.

2.3.4 Improvement 4: Dependency Based Negative Event Generation

Instead of using a window-based trace comparison algorithm, an alternative way of generating artificial negative events is to directly use the discovered structural information from the temporal frequent constraints and association rules. Locality, (long-distance) dependency and implicit dependency information can thus be applied towards the induction of negative events.

2.3.4.1 Locality (or: Explicit Dependencies)

It is possible to generate negative events based on locality information (i.e. explicit dependencies). As a rule of thumb, negative events with a certain activity type can be added before a given completed event in a process instance when this activity type is not locally dependent on the activity type of the event completed at the previous position:

$$N := N \cup (\sigma, i, a) \impliedby \forall a \in A_L, \sigma \in L : [\forall \sigma_i \in \sigma, \sigma_i \neq a : [\sim \mathtt{Local}(\sigma_{i-1}, a)]]$$

Instead of using $\mathtt{ChainPrec}(a,c) \vee \mathtt{ChainResp}(a,c)$ in the derivation rule for $\mathtt{Local}$, we can also define a $\mathtt{StrongLocal}$ variant using $\mathtt{ChainPrec}(a,c) \wedge$

$\mathtt{ChainResp}(a,c)$ instead. Using $\mathtt{StrongLocal}$ instead of $\mathtt{Local}$ in the rule above leads to a weaker check before generating a negative event, since, in most cases, less $\mathtt{StrongLocal}$ constructs can be discovered than $\mathtt{Local}$. Recall also that the construction of $\mathtt{Local}$ predicates takes parallelism information into account.

2.3.4.2 Long-Distance (with: Implicit) Dependencies

We define the following structural derivation rules to mine all dependencies between activity types (both implicit and explicit), similar to $\mathtt{Local}$ (explicit dependencies only):

$$\begin{aligned}\forall a,b \in A_L :& (\mathtt{Precedence}(a,b) \vee \mathtt{Response}(a,b)) \wedge \sim \mathtt{Parallel}(a,b)\\ &\implies \mathtt{Dependence}(a,b)\\ \forall a,b \in A_L :& (\mathtt{Precedence}(a,b) \wedge \mathtt{Response}(a,b)) \wedge \sim \mathtt{Parallel}(a,b)\\ &\implies \mathtt{StrongDependence}(a,b)\end{aligned}$$

Based on these dependencies, the rule of thumb defined above can be expanded. Even negative events with a certain activity type that is locally dependent on the activity type of the event completed at the previous position can be added before a given completed event, so long as not *all* activities on which the negative event under consideration is *strongly* dependent ($\mathtt{StrongDependence}$) on were completed before, or so long as *no single* activity on which the negative event under consideration is dependent on ($\mathtt{Dependence}$) has completed before.

$$\begin{aligned}N := N \cup (\sigma,i,a) \impliedby & \forall \sigma \in L, a \in A_L : [\forall \sigma_i \in \sigma, \sigma_i \neq a : [\\ & \exists b \in A_L, b \notin \langle \sigma_1, \ldots, \sigma_{i-1} \rangle : [\\ & \quad \mathtt{StrongDependence}(b,a)]]]\end{aligned}$$

$$\begin{aligned}N := N \cup (\sigma,i,a) \impliedby & \forall \sigma \in L, a \in A_L : [\forall \sigma_i \in \sigma, \sigma_i \neq a : [\\ & \nexists b \in A_L, b \in \langle \sigma_1, \ldots, \sigma_{i-1} \rangle : [\\ & \quad \mathtt{Dependence}(b,a)]]]\end{aligned}$$

Finally, we can also use the $\mathtt{StrongDependence}$ construct to suggest an optimal minimum window size, i.e. a window size which is able to capture pairs of activity types between which an implicit (long-distance) dependency relation exists. To do so, we need to restrict $\mathtt{StrongDependence}$ a bit further to drop strong dependencies which do not correspond with an implicit dependency in the underlying process model. For example, a $\mathtt{StrongDependence}$ construct can always be found between starting and ending activities. However, a starting activity is always followed by the ending activity, so that this dependency is not an implicit one. Deriving a window size from this construct would lead to a useless suggestion, as this would give the same result as when using an unlimited window. Therefore, we restrict our search to unique strong dependencies (derived with the $\mathtt{UniqueStrongDependence}$ rule below) where activity types are strongly dependent on one other activity type only, which is a good indication for the presence of an implicit dependency.

$$\begin{aligned}&\forall a, b \in A_L : [\mathit{StrongDependence}(a, b) \wedge \nexists c \in A_L, c \neq a : [\\ &\quad \mathit{StrongDependence}(c, b)]] \implies \mathit{UniqueStrongDependence}(a, b)\end{aligned}$$

A suggestion towards a window size can then be given by:

$$\begin{aligned}ws_{suggested} :=& \mathit{Max}_{a,b \in A_L : \mathit{UniqueStrongDependence}(a,b)}(\\ &\mathit{Min}_{\sigma \in L, \sigma_i, \sigma_j \in \sigma : [\sigma_i = a, \sigma_j = b, i<j]}(i - j))\end{aligned}$$

Note that, when sequences are present in the event log which contain multiple events corresponding to the same activity type (e.g. for process models containing loops), $(i-j)$ is computed such that the resulting difference between the two events is minimal but greater than zero.

2.4 Experimental Results

We have implemented our revised artificial negative event generation technique in ProM 6. We test the improvements above with the *driversLicenseLoop* event log, an artificial process log which has been used before by Alves de Medeiros et al. [66] to evaluate the Genetic Miner discovery algorithm, and by Goedertier et al. [2] as a testcase for AGNEsMiner. The *driversLicenseLoop* process is interesting, since it contains parallelism, recurrence, duplicate tasks and implicit dependencies. To compare the different settings of the artificial event generation algorithm, a process log containing 350 process instances was used ($|L| = 350$, $|\bigcup L| = 87$, $|A_L| = 11$). Figure 2.7 depicts the underlying Petri net for the *driversLicenseLoop* process, for illustrative purposes.

Table 2.1 lists the various parameter configurations used to evaluate the artificial event generation technique. For each of the configurations, all generated

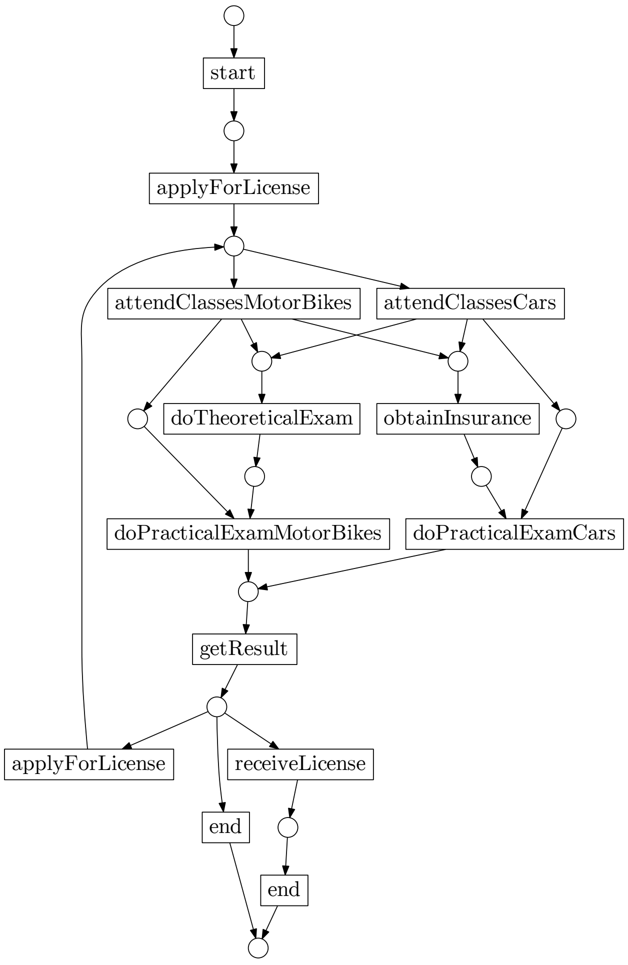


Figure 2.7: The *driverseLicenseLoop* process.

Table 2.1: Used parameter setting configurations for the artificial event generation tests.

Parameter Configuration Identifier	Window Generation	Window Size	Dynamic Window	Minimum Window Size	Structural Generation
originalWs-1	yes	-1	no	–	no
originalWs3	yes	3	no	–	no
originalWs1	yes	1	no	–	no
dynamicWs-1MinWs-1	yes	-1	yes	-1	no
dynamicWs3MinWs-1	yes	3	yes	-1	no
dynamicWs1MinWs-1	yes	1	yes	-1	no
dynamicWs-1MinWs1	yes	-1	yes	1	no
dynamicWs3MinWs1	yes	3	yes	1	no
dynamicWs1MinWs1	yes	1	yes	1	no
strucOnly	no	–	–	–	yes
strucNonDynamicWs-1	yes	-1	no	–	yes
strucNonDynamicWs3	yes	3	no	–	yes
strucNonDynamicWs1	yes	1	no	–	yes
strucDynamicWs-1MinWs-1	yes	-1	yes	-1	yes
strucDynamicWs3MinWs-1	yes	3	yes	-1	yes
strucDynamicWs1MinWs-1	yes	1	yes	-1	yes
strucDynamicWs-1MinWs1	yes	-1	yes	1	yes
strucDynamicWs3MinWs1	yes	3	yes	1	yes
strucDynamicWs1MinWs1	yes	1	yes	1	yes

negative events were introduced in the given event log. The "Window Generation" parameter denotes the use of window based artificial negative event generation, "Window Size" denotes the size of the window, "Dynamic Window" denotes the use of the dynamic window improvement with a minimum required window size "Minimum Window Size". "Structural Generation" defines if dependency based artificial negative event generation is performed, either on its own ("strucOnly"), or together with window based negative event generation ("strucNonDynamicWs-1" and following are obtained by merging a window based generated set of negative events with the set of negative events obtained from "strucOnly"). We use window sizes -1 (unlimited), 3 (suggested by $ws_{suggested}$) and 1 (most limited) to test the event generation procedure. When a dynamic window is used, we use both -1 (candidate disprove window must be as long as window in positive trace) and 1 (no effective minimum) as required minimum window values. "Original" configuration identifiers correspond with a parameter setting which could be obtained with the original version (i.e., no improvements) of the artificial event generation algorithm.

Table 2.2 gives the results for each of the above defined parameter setting configurations. We compare the results for the various parameter configurations with two given sets of negative events. A "naive generation method" constructs a set of negative events by injecting at each position in a trace a negative event for each activity type, except the activity type equal to the (completed) event at the current position, i.e. $\forall \sigma \in L : [\forall \sigma_i \in \sigma : [\forall a \in A_L \setminus \{\sigma_i\} : [N_n := N_n \cup (\sigma, i, a)]]]$.

Note that even when this naive method is used, the number of incorrect negative events in respect to the total negative events is rather low. This is in fact a good indication towards the fact that the given event log gives a good coverage of all possible execution traces as allowed by the underlying process model. The "Fully Correct Log" was constructed based on the given Petri net used to simulate the drivers license process (N_c). Of course, in real-life cases, such a reference model is unavailable, preventing the construction of a fully correct set of negative events by which the artificial induction results can be evaluated. For each parameter configuration, we calculate the correctness and completeness ratio. Correctness is defined as one minus the ratio of incorrect negative events to the number of incorrect negative events generated by the naive method. Completeness is defined as the ratio of correct negative events over the full number of possible, correct negative events, as given by the fully correct log, i.e. $correctness = 1 - \frac{N^{incorrect}}{N_n^{incorrect}}$ and $completeness = \frac{N^{correct}}{N_c^{correct}}$.

The following conclusions can be derived from the results. First, the inherent trade-off between correctness and completeness becomes apparent here, as most configurations show an inverse relation between the two requirements. Second, we note that no single window size configuration is able to generate a set of negative events which is both correct and complete when using the original version of the artificial event generation algorithm. Next, using a strict window size (1) in combination with the dynamic window improvement leads to a set of negative events which is fully correct, albeit not complete. Constructing a set of negative events which is both complete and correct is possible if the window size is increased to 3 (suggested by investigating the structure of implicit dependencies—denoted in bold case in Table 2.2), or by using non-window dependency based generation, which also leads to an acceptable completeness value (98%). Moreover, using dependency-based generation ensures the addition of "non-trivial" negative events, derived from implicit dependencies, which proves especially helpful in a later phase when the set of negative events is used for evaluation or discovery tasks. The results also deal with another concern: even although we have defined a large number of parameters, two straightforward, well performing defaults can be suggested: either apply window based generation with a dynamic window of size 1 in conjunction with dependency based event generation, or only apply window based generation with a window size equal to the suggested window size.

Table 2.2: Results of the *driversLicenseLoop* experiment under various configurations.

Parameter Configuration Identifier	Incorrect Negative Events	Total Negative Events	Correctness	Completeness
Fully Correct Log	0	44866	100%	100%
Naive Generation Method	2484	47350	0%	100%
originalWs-1	642	45508	74,2%	100%
originalWs3	621	45487	75,0%	100%
originalWs1	621	44866	75,0%	98,6%
dynamicWs-1MinWs-1	642	45508	74,2%	100%
dynamicWs3MinWs-1	0	44866	**100%**	**100%**
dynamicWs1MinWs-1	0	42382	100%	94,5%
dynamicWs-1MinWs1	642	45508	74,2%	100%
dynamicWs3MinWs1	0	44866	**100%**	**100%**
dynamicWs1MinWs1	0	42382	100%	94,5%
strucOnly	0	40469	100%	90,2%
strucNonDynamicWs-1	642	45508	74,2%	100%
strucNonDynamicWs3	621	45487	75,0%	100%
strucNonDynamicWs1	621	45349	75,0%	99,7%
strucDynamicWs-1MinWs-1	642	45508	75,2%	100%
strucDynamicWs3MinWs-1	0	44866	**100%**	**100%**
strucDynamicWs1MinWs-1	0	43969	100%	98,0%
strucDynamicWs-1MinWs1	642	45508	74,2%	100%
strucDynamicWs3MinWs1	0	44866	**100%**	**100%**
strucDynamicWs1MinWs1	0	43969	100%	98,0%

2.5 Conclusions

Process mining analysis tasks are often confronted with some particular difficulties. One such difficulty is that process mining is commonly limited to the harder setting of unsupervised learning, since negative information about state transitions that were prevented from taking place (i.e. negative events) is often unavailable in real-life event logs and consequently cannot guide the learning tasks. In this chapter, we have outlined a first extension for the negative event generation method as first introduced in the AGNEsMiner process discovery algorithm [2]. Generating a robust set of negative events boils down to finding an optimal set of negative examples under the counteracting objectives of correctness and completeness. Correctness implies that the generation of false negative events has to be prevented, while completeness entails the induction of "non-trivial" negative events, that is, negative events which are based on constraints imposed by complex structural behavior, such as non-local, history-dependent constructs. The existence of the tradeoff between these two goals is due to the completeness assumption made over an event log when generating artificial negative events. In the original version of the algorithm, a configurable completeness assumption is defined by proposing a window size parameter and a negative event injection probability. We presented several improvements to make this configured completeness assumption less strict and the artificial event generation procedure more robust in order to prevent the introduction of falsely induced negative events in cases where an event log does not capture all possible behavior. Ensuring a correct induction of negative events proves especially helpful in practice when these events are subsequently used for evaluation purposes. The improvements entail: firstly, new definitions for temporal constraint confidence measures; second, a variant calculation technique based on recurrent behavior (short loops); third, a redefinition of the window based trace similarity check; fourth, the definition and appliance of a negative event score value which makes the tradeoff between completeness and correctness explicit; finally, the appliance of non-window based negative event generation techniques, based on discovered dependency information. By means of an experiment on an artificial log, we illustrate the benefits of the proposed improvements.

This first approach already offers a more robust method towards inducing artificial negative events, although some problem still remain. First, the induction procedure is slow, due to the fact that the whole (distinct) event log is iterated upon, the time-consuming steps of deriving temporal and structural information, and the exponential complexity nature of parallel and recurrent variant generation. Second, end-users are required to supply a number of parameters to drive the induction process, forcing the make an assumption about the degree of completeness of the given event log, which is not always known up-front. Third, negative events are induced or not, whereas it would be better to assign some degree of "confidence" to their existence.

In the next chapter, we will set forth towards solving these issues, as well as developing an application of artificial negative events in the form of a comprehensive conformance checking framework.

Chapter 3

Conformance Checking with Weighted Artificial Negative Events

"A foolish consistency is the hobgoblin of little minds."
– Ralph Waldo Emerson

3.1 Introduction

In this chapter, we will develop a novel conformance checking framework based on the concept of artificial negative events. Recall from the introductory chapter that conformance checking pertains to the mining task where existing process models are compared with the real-life behavior as captured in event logs so as to measure how well a process model performs with respect to the actual executions of the process at hand [100]. As such, the "goodness" of a process model is typically assessed over the following four quality dimensions [47, 124]: fitness (or: recall, sensitivity), indicating the ability of the process model to correctly replay the observed behavior; precision (or: appropriateness), i.e. the model's ability to disallow unwanted behavior; generalization, which indicates the model's ability to avoid overfitting; and finally, simplicity (or: structure, complexity, comprehensibility), stating that simpler process models should be preferred above more complex ones if they are able to fit the observed behavior just as well, thus embodying the principle of Occam's Razor.

Our approach offers the following contributions. First, we develop and apply a significantly improved artificial negative induction strategy, based on the approach developed in the previous chapter, but extending it with a novel weighting method which tackles the problems of scalability and the requirement of end-user knowledge regarding the level of log completeness. Second, the concept of weighted artificial negative events is used as the basis for two new conformance checking metrics: *Weighted Behavioral Precision* and *Weighted Behavioral Generalization*. Most existing literature has focused on the fitness and precision of process models, whereas the ability to generalize has been much more difficult to estimate or describe. Our method is able to assess both precision and generalization, taking into account the inherent tradeoff between these two dimensions in an explicit manner for the evaluation of process models. Third, we discuss in depth the problem of replaying event sequences on Petri nets for sequences containing both positive and negative events. We also identify some important remarks concerning Petri net replay methods when checking precision and generalization. Fourth and finally, all described algorithms were implemented in a number of ProM plugins; a Petri net conformance checking tool was developed to inspect model quality, conformance and deviations in a visual manner.

3.2 Preliminaries

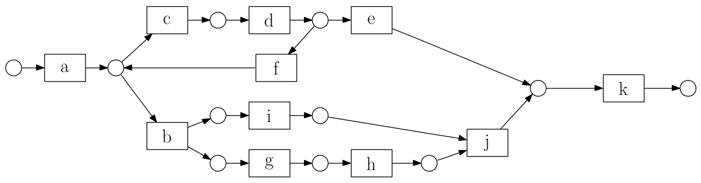
This chapter will use the definition of an event log, Petri net, and Petri net execution semantics as given by the introductory chapter. Recall in particular our comment regarding the construction of a firing sequence through transitions in a Petri net to match a given trace from an event log. We have stated that it is possible that multiple firing sequences can be found to replay a trace. When a firing sequence can be found for a trace so that all transitions can be fired sequentially without one of them being forced to fire, it is said that the trace fits in the Petri net (or: that the trace can be replayed correctly by the Petri net). Note that replay is closely related to the first model quality dimension: fitness. That is, Petri nets where a higher amount of traces contained in an event log can be parsed by the net are preferred above nets which are less able to replay the traces without force firing transitions. For now, we will make abstraction of the exact manner by which a firing sequence can be constructed through a Petri net for a given trace. It suffices to assume that replaying a trace on a Petri net will result in a single "best fitting" firing sequence: that is, the firing sequence containing the least amount of force fired transitions (preferably none) and the least amount of invisible transitions. Section 3.6.1 provides a thorough discussion on how such a firing sequence can be computed.

Figure 3.1 depicts an event log together with a set of process models to illustrate the impact of the four quality dimensions, similarly as done in [124]:

- *perfectModel* shows a high quality, "perfect" model: all traces contained in the event log can be replayed by the model (fitness), the model does not allow for extra behavior not found in the event log (precision), the model does not overfit the event log (generalization) and is structurally simple and easy to understand (simplicity).
- *singleModel* represents only one "single path" from start to finish. As such, the model is very simple, has a high precision, but is not able to generalize. In addition, only a small subset of traces can be replayed without force firing transitions, so that the fitness of this model is low as well.
- *flowerModel* permits any sequence of transitions (between starting and ending transitions). As such, flower models are very simple, generalize well and are able to fit all traces contained in the event log, but score low on precision since a lot of additional behavioral not found in the event log is allowed.
- *connectedModel* is comparable to a flower model with regards to the four quality dimensions, except that here, a fully connected Petri net is used with an abundance of invisible "routing" transitions in order to allow any sequence of activities[1], making the model much harder to understand.
- *stackedModel* shows a model where each trace in the event log is modeled as a separate path of transitions between the start and end place (remark here the occurrence of duplicate transitions). Although well performing in terms of fitness and precision, it is clear that this model heavily overfits the event log and is thus not able to generalize. In addition, the model is not simple to interpret. Although the structural logic behind a stacked model looks simple enough, finding out which path should be followed in order to replay a trace is more difficult, especially during execution where the future behavior of a process instance is very likely to be unknown at the current point in time.

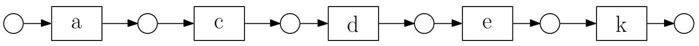
[1]The fully connected model may appear rather unrealistic. However, note that Causal nets [43], another representational form for process models, may end up looking like *connectedModel* after converting such models to Petri nets.

Multiplicity of σ	Trace $\sigma \in L$
113	$\langle a, c, d, e, k \rangle$
110	$\langle a, b, i, g, h, j, k \rangle$
74	$\langle a, b, g, i, h, j, k \rangle$
63	$\langle a, b, g, h, i, j, k \rangle$
39	$\langle a, c, d, f, c, d, e, k \rangle$
30	$\langle a, c, d, f, b, i, g, h, j, k \rangle$
19	$\langle a, c, d, f, b, g, i, h, j, k \rangle$
16	$\langle a, c, d, f, b, g, h, i, j, k \rangle$
8	$\langle a, c, d, f, c, d, f, b, g, h, i, j, k \rangle$
8	$\langle a, c, d, f, c, d, f, c, d, e, k \rangle$
8	$\langle a, c, d, f, c, d, f, b, i, g, h, j, k \rangle$
5	$\langle a, c, d, f, c, d, f, b, g, i, h, j, k \rangle$
3	$\langle a, c, d, f, c, d, f, c, d, f, b, i, g, h, j, k \rangle$
2	$\langle a, c, d, f, c, d, f, c, d, f, c, d, e, k \rangle$
2	$\langle a, c, d, f, c, d, f, c, d, f, b, g, h, i, j, k \rangle$
$\lvert L \rvert = 500$	
$\lvert \bigcup L \rvert = 15$	



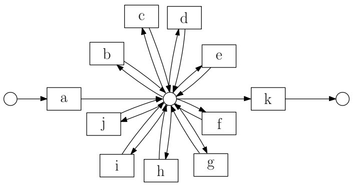
perfectModel: Perfect Model
Fitness +, Precision +, Generalization +, Simplicity +

Figure 3.1: An event log *exampleLog* together with five process models. The process models illustrate the impact of the four quality dimensions: fitness, precision, generalization and simplicity. Here, the event log and the *perfectModel* are shown.



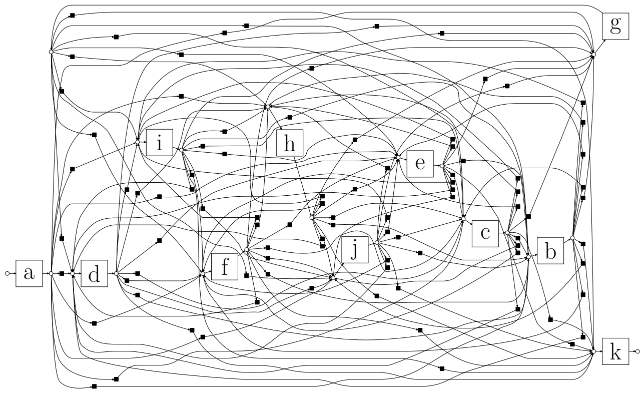
singleModel: Single Path Model
Fitness -, Precision +, Generalization -, Simplicity +

Figure 3.1 (continued): The *singleModel* process model.



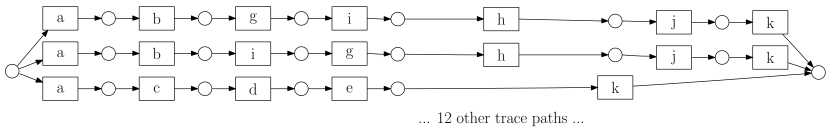
flowerModel: Flower Model
Fitness +, Precision -, Generalization +, Simplicity +

Figure 3.1 (continued): The *flowerModel* process model.



connectedModel: Fully Connected Model
Fitness +, Precision -, Generalization +, Simplicity -

Figure 3.1 (continued): The *connectedModel* process model.



stackedModel: Single Path Model
Fitness +, Precision +, Generalization -, Simplicity -

Figure 3.1 (continued): The *stackedModel* process model.

3.3 Weighted Artificial Negative Events

Our precision and generalization evaluation approach is based on the use of weighted artificial negative events. Weighted artificial negative events extend the concept of artificial negative events with a scoring mechanism in order to make the evaluation of process models more robust as event logs become less complete.

3.3.1 Rationale and Formalization

In the previous chapter, an improved artificial negative event generation strategy was proposed which allows to configure the completeness assumption made over a given event log in a more robust and fine-grained manner. However, we have stated that this approach still exhibits the following problems:

- Scalability. The induction procedure is slow, due to the fact that the whole (distinct) event log is iterated upon, the time-consuming steps of deriving temporal and structural information, and the exponential complexity nature of parallel and recurrent variant generation.
- Domain knowledge. Even although the completeness assumption was made more configurable, end-users were still required to configure the induction process.
- Binary induction. Even when taking into account existing techniques to estimate the completeness of a given event log without an a-priori known process model [125, 126], which could be applied in order to guide the artificial negative event generation algorithm, the problem still remains

Algorithm 3.1 Weighted artificial negative event generation algorithm.

```
Input: An event log $L$
Output: A set $N$ of induced artificial negative events
 1: function NE($\sigma_i$) % Induce set of negative events
 2:     $N := \emptyset$
 3:     for all $a \in A_L \setminus \{\sigma_i\}$ do
 4:         $s := 1$ % Score for this negative event
 5:         for all $\upsilon \in L$ do
 6:             for all $\upsilon_j \in \upsilon | \upsilon_j = a$ do
 7:                 % Calculate unmatching window ratio
 8:                 $ws := i - 1$ % Window size
 9:                 $mws := 0$ % Matching window size
10:                 $l := 1$
11:                 while $l < Min(i, j) - 1 \land \sigma_{i-l} = \upsilon_{j-l}$ do
12:                     $mws := mws + 1$
13:                     $l := l + 1$
14:                 end while
15:                 $uwr := \frac{ws - mws}{ws}$ % Matching window ratio for this comparison
16:                 $s := Min(s, uwr)$
17:             end for
18:         end for
19:         % Negative event with activity $a$ and weight $s$
20:         $N := N \cup \{(\sigma, i, a, s)\}$
21:     end for
22:     return $N$
23: end function
```

that negative events would be either induced or not, without some kind of strength or confidence associated to their presence based on the structure contained in the given log, which can be considered as being too coarse-grained.

To resolve these issues, we propose a scoring method which can be used to weight negative events in terms of their confidence. That is, the higher the weighting of a negative event, the less likely it is deemed that this negative event will be refuted by additional traces as generated by the real underlying, unknown process and vice versa[2]. The calculation of this weight is done in the following manner (formalized as a definition for function $NE(\sigma_i)$ in Algorithm 3.1): we calculate the score for a negative event with activity $a \in A_L \setminus \{\sigma_i\}$. For each trace υ containing activity a at υ_j, the event window before υ_j is compared with the event window before σ_i in the original trace σ in order to obtain the "unmatching window ratio", i.e. the length of the unmatching window divided by the total window length in σ, working backwards from σ_i and υ_j: $\frac{|window| - |matching\ window|}{|window|}$. Remark that it is possible for a single trace υ to contain multiple activities equal to the negative event under consideration, so that one trace can give rise to multiple comparisons, and thus different unmatching window ratios. Remark also that the current $\sigma \in L$ itself may also contain the activity for the candidate negative event

[2]"Non numerantur, sed ponclerantur," as said by the great Paul Erdős.

under consideration. Finally, to obtain the final weighting for a negative event, the minimum unmatching window ratio is taken over all comparisons performed (line 16): $\mathtt{Min}_{\forall \textit{window comparisons}}(\frac{|\textit{window}|-|\textit{matching window}|}{|\textit{window}|})$. To induce all weighted artificial negative events, function $\mathtt{NE}(\sigma_i)$ is called for each $\sigma_i \in \sigma, \sigma \in L$.

The weighting for each negative event can be summarized as follows: the *longer a matching prefix* can be found equal to the prefix before the negative event under consideration, the *smaller the unmatching window* will become and a *lower weight* will be given to the negative event. A *weighting of 0* (minimum) indicates that a trace was found *containing the same full prefix* as seen before the negative event under consideration, indicating that this behavior in fact did occur and as such cannot be supported at all as being disallowed (i.e. negative) behavior. A *weighting of 1* (maximum) indicates that there did *not exist any trace where the candidate negative event's activity occurred and was preceded by a matching prefix* (even of length 1). Strong evidence then exists that the behavior represented by the candidate negative event should indeed be disallowed.

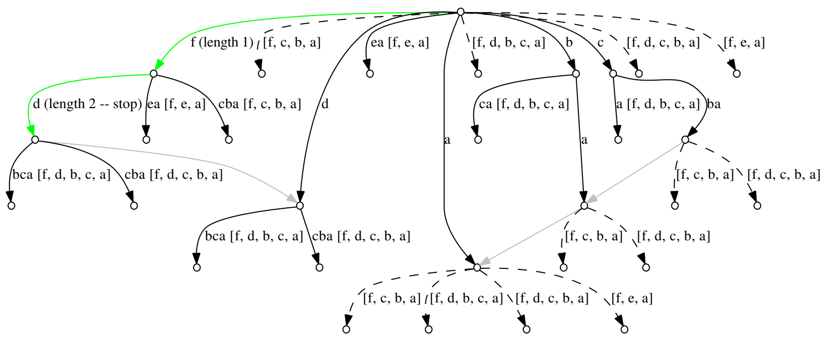
An example can help to clarify the weighting procedure:

σ		1	2	3	**4**	5
		a	b	c	**(y?) x**	d
υ	1	2	3	4	**5**	6 .
	e	a	f	c	**y**	g
window:	–	–	$\neq$	$=$	$\leftarrow$	

Consider the trace $\sigma \in L = \langle a, b, c, x, d \rangle$; we wish to obtain the unmatching window ratio for a negative event, say y, inserted before $\sigma_4 = x$ by comparing the window before σ_4 (i.e.: $\langle a, b, c \rangle$, $|\textit{window}| = 3$) with the window before υ_5 in the trace $\upsilon = \langle e, a, f, c, y, g \rangle$, which is another trace in L that does contain event y. The window in υ is thus: $\langle e, a, f, c \rangle$. These two windows ($\langle a, b, c \rangle$ and $\langle e, a, f, c \rangle$) are now compared as follows. We calculate the matching window length, starting backwards from both windows. The first pair of events, $\sigma_3 = \upsilon_4 = c$, are indeed equal, so the matching window length is incremented with 1. The next pair, $(\sigma_2 = b) \neq (\upsilon_3 = f)$, however, is not equal, so the comparison is ended at this point, even though the next pair $\sigma_1 = \upsilon_2 = a$ is equal again. The unmatching window ratio for this result thus amounts to $\frac{3-1}{3} = 0.66$. The final weighting for this negative event is obtained by taking the minimum unmatching window ratio over all similar window comparisons which could be performed.

3.3.2 Scalability

The scalability of the weighted negative artificial event induction procedure as described above is weak. The complexity of inducing *all* weighted artificial negative events in an event log L with Algorithm 3.1 can be expressed as follows: $O((|L| \times |\mu|) \times (|A_L|) \times (|L| \times |\mu| \times |\mu|))$. That is, for each trace in the event log L, and for each position in that trace ($|\mu|$ is equal to the length of the longest trace in the event log L and forms an upper bound for the number of positions iterated), function $NE(\sigma_i)$ is evaluated to induce the negative events for this position, which considers each activity in A_L as a candidate negative event. For each such candidate event, the traces in the event log are iterated again together with its activities, and a window comparison is performed whenever an activity is encountered which is equal to the current candidate negative event (worst case meaning every position in every trace with a maximum window length of $|\mu|$). This evaluates to $O(|L|^2 \times |\mu|^3 \times |A_L|)$. To deal with the problem of scalability, we utilize Ukkonen's algorithm [127] in order to construct a suffix tree over the event log in order to quickly perform window lookups. As an example, consider the trace $\langle a, b, c, d, e, f, g\rangle$. A suffix tree over this trace yields the following retrievable suffixes: $\langle a, b, c, d, e, f, g\rangle$ (the trace itself), $\langle b, c, d, e, f, g\rangle$, $\langle c, d, e, f, g\rangle$, $\langle d, e, f, g\rangle$, $\langle e, f, g\rangle$, $\langle f, g\rangle$ and $\langle g\rangle$. Suppose now we want to find the matching length for the window $\langle x, c, d\rangle$ (i.e. 2). As it stands, the suffix tree does not contain an entry point for x and as such, the matching length is deemed to be 0. One solution is to iterate over the trace and add the suffixes of $\langle a, b, c, d, e, f, g\rangle, \langle b, c, d, e, f, g\rangle, \ldots, \langle g\rangle$, but the more optimal and preferred approach is to construct the suffix tree over the reversed trace, i.e. $\langle g, f, e, d, c, b, a\rangle$ resulting in the following suffixes: $\langle g, f, e, d, c, b, a\rangle$, $\langle f, e, d, c, b, a\rangle$, $\langle e, d, c, b, a\rangle$, $\langle d, c, b, a\rangle$, $\langle c, b, a\rangle$, $\langle b, a\rangle$ and $\langle a\rangle$. To find the matching window length of $\langle x, c, d\rangle$, the window is traversed backwards so that now, $\langle d, c\rangle$ is found with a length of 2. Observe that this reversal has the effect of losing the online property of Ukkonen's algorithm, but retains linear-time construction complexity. The complexity of the implemented weighted artificial negative event generation algorithm to generate all weighted artificial negative events in an event log is thus as follows: $O((|L| \times |\mu|) + (|L| \times |\mu|) \times (|A_L|) \times (|\mu|))$; the suffix tree construction step is executed once and is linear in the size of the alphabet (in this case, the length of the longest trace times the log size is an upper bound) [127]. Next, every trace in the event log is iterated once more at each position, and each activity is still evaluated as a candidate negative event. However, the lookup of the matching window now scales linearly with the length of the longest trace. As such, the overall complexity evaluates to $O(|L| \times |\mu|^2 \times |A_L|)$, so that the induc-



Event log $L = \{\langle a, b, c, d, f\rangle, \langle a, c, b, d, f\rangle, \langle a, b, c, f\rangle, \langle a, e, f\rangle\}$

Figure 3.2: Illustration of a suffix tree built over an event log L. We illustrate the lookup of a window $\langle x, d, f\rangle$. The window is iterated from right-to-left. Activities f and d are found (green arrows on the left—dark gray in gray scale), but no edge to x exists, so that the matching window length evaluates to 2 for this comparison.

tion algorithm now scales linearly with the size of the event log and the activity alphabet. To illustrate the workings of the suffix tree, Figure 3.2 provides a visual example of a suffix tree built over a small event log, and how a window lookup is performed in practice.

This contribution regarding the weighting of negative events and the application of suffix trees greatly improves the robustness of the negative event induction to varying levels of event log completeness and the time needed to generate artificial negative events compared to existing techniques, especially since the trace variant generation step can be dropped without a significant loss of accuracy in the weighting of a negative event, as will be shown in the following subsection.

3.3.3 Empirical Validation

To validate our weighted artificial negative event approach, we apply the proposed generation technique (Algorithm 3.1) on a number of process event logs in a controlled environment for which the reference model is known beforehand. Having such a reference model allows us to construct a complete and correct set of negative events, as defined by the process model itself, by which the artificially generated set of events can then be evaluated. Note that, in real-life process mining settings, a true reference model is, of course, almost always unavailable, so that these models are only applied in this section as a means to validate the performance of the weighted artificial negative event generation procedure.

The procedure to generate negative events in a given log based on a (fitting) Petri net is as follows. Just as before, for each trace $\sigma \in L$, we check at each position σ_i which negative events can be induced. To do so, a new trace τ is constructed, consisting of the full prefix before (not including) σ_i in σ, and with the candidate negative activity a appended at the end, i.e. $\tau = \langle \sigma_1, \ldots, \sigma_{i-1}, a \rangle$. The reference model is then queried to see whether it was possible for the negative event under consideration to occur instead of σ_i, by investigating whether a fitting firing sequence $f^{M_0,(P,T,F)}$ can be found which can be mapped to τ using μ and which contains no forced transitions. If such a fitting firing sequence cannot be found, the candidate negative event can be inserted in τ before τ_i[3]. Algorithm 3.2 describes this approach in a formalized manner.

Algorithm 3.2 Generating artificial negative events from a Petri net in an event log.

```
Input:  An event log $L$
Input:  $(P, T, F), M_0$ % Given Petri net and initial marking
Input:  $\mu$ % Transition-activity mapping $\mu : T \mapsto A_L \cup \{a_i, a_b\}$
Output:  A set $N$ of induced artificial negative events
 1: function NEP($\sigma_i$) % Induce set of negative events from Petri net
 2:     $N := \emptyset$
 3:     for all $a \in A_L \setminus \{\sigma_i\}$ do
 4:         $s := 1$ % Score for this negative event
 5:         $\tau := \langle \sigma_1, \ldots, \sigma_{i-1}, a \rangle$
 6:         % The $CanReplay$ function returns a true or false value depending on whether the given Petri
    net is able to replay a given trace starting from the initial marking under a given mapping in a fitting-way,
    i.e. without any force firings
 7:         if $CanReplay((P, T, F), M_0, \mu, \tau)$ then
 8:             $s := 0$ % The Petri net can replay the “negative trace”, so the candidate negative event is not a
    true one
 9:         end if
10:         % Negative event with activity $a$ and weight $s$
11:         $N := N \cup \{(\sigma, i, a, s)\}$
12:     end for
13:     return $N$
14: end function
```

Twenty-five different event logs, containing a variety of structural constructs and differing in complexity, have been utilized. Twenty of these logs were used before by Alves de Medeiros et al. [66] and have since become part of a widely used benchmarking set in the field of process mining. Additionally, another log, *complex*, was built, containing complex behavior (e.g. nested loops and parallelism). Table 3.1 lists the main characteristics of these event logs. The overview

[3]As a note, the observant reader might be wondering why not a firing sequence is constructed for $\tau = \langle \sigma_i \rangle$ for $i = 1, \ldots, i-1$ which is then queried for all disabled transitions—which would provide information for all negative events immediately at σ_i, instead of having to query all $a \in A_L \setminus \{\sigma_i\}$ separately. The reason why this is done because a transition mapped to a might not be enabled in this final state, but there might exist enabled invisible transitions which can be fired to subsequently enable such a transition. Since such a firing sequence would be mapped to the same resulting event log trace, we need to consider these cases as well and thus check whether we can find a fitting firing sequence for the full $\tau = \langle \sigma_i, a \rangle$ for $i = 1, \ldots, i-1$ sequence.

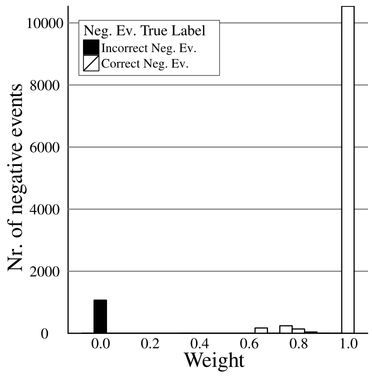
Table 3.1: Characteristics of event logs and reference models used in the weighted artificial negative event validation setup.

Event Log	$\lvert A_L \rvert$	$\lvert L \rvert$	$\lvert \bigcup L \rvert$	parallelism?	loops?	invisibles?	non-free choice?	duplicates?
a10skip	12	300	6	✓		✓		
a12	14	300	5	✓				
a5	7	300	13	✓	✓	✓		
a6nfc	8	300	3	✓		✓	✓	
a7	9	300	14	✓				
a8	10	300	4	✓				
betaSimplified	13	300	4			✓	✓	✓
choice	12	300	16			✓		
driversLicense	9	2	2			✓		
driversLicenseLoop	11	350	87	✓	✓	✓	✓	✓
herbstFig3p4	12	32	32	✓	✓			
herbstFig5p19	8	300	6	✓		✓		✓
herbstFig6p18	7	300	153		✓	✓		
herbstFig6p31	9	300	4					✓
herbstFig6p36	12	300	2			✓	✓	
herbstFig6p38	7	300	5	✓				✓
herbstFig6p41	16	300	12	✓				
l2l	6	300	10		✓	✓		
l2lOptional	6	300	9		✓	✓		
l2lSkip	6	300	8		✓	✓		
complex	19	6107	1006	✓	✓			✓
hospital (real-life)	626	1143	981	–	–	–	–	–
incident (real-life)	18	24770	1174	–	–	–	–	–
telecom (real-life)	42	17812	1908	–	–	–	–	–
ticketing (real-life)	9	276599	3140	–	–	–	–	–

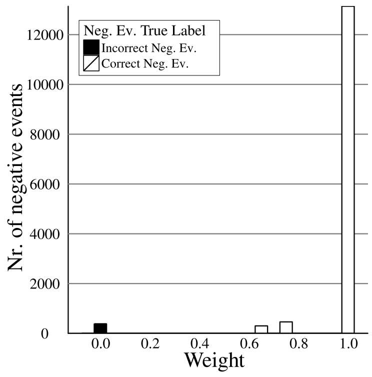
also includes four real-life logs which will be used in further sections.

3.3.3.1 Correctness

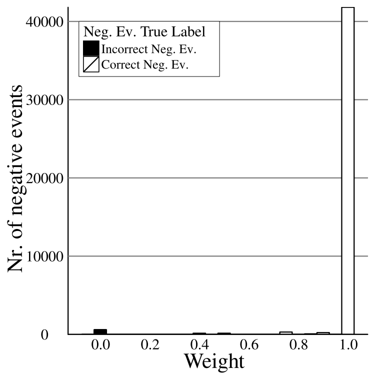
The first validation task investigates whether the weight given to generated artificial negative events is able to correctly differentiate between correct and incorrect negative events, as indicated by the reference model. For a well-performing weighting metric, we would expect that the majority of true, correct negative events is given a high weight, whereas false, incorrect negative events are assigned a low weight. Our tests indeed show that this is the case for the logs included in our experiment. Figure 3.3 depicts the distribution of artificial negative event weights for ten logs contained in the test set (the distributions for the other logs are similar and omitted for brevity). For real-life logs *incident* and *telecom*, no reference model is available, so that only the distribution of generated negative event weights are given (colored gray) for the sake of completeness. The distributions show that, for most candidate negative events, the weighting technique is able to correctly and unambiguously (i.e. with absolute confidence) indicate whether the negative event is valid (weight of 1) or invalid (weight of 0). As logs become more complex or less complete, more candidate negative events will occur for which it is impossible (based on the given log) to unambiguously derive whether these candidates are valid or not. They are given a weight between 0 and 1. Our experiment shows that these weights correspond with our initial requirement, namely the assignment of a lower weight to incorrect negative events (as determined by the reference model available in this controlled setup) and a higher weight to correct negative events. While it is the case that the correctness of the multitude of negative events can be determined in a straightforward manner based on the given log (i.e. all negative events with weight equaling 1), it should be noted that the negative examples which cannot be derived as easily often reveal the most discerning information, such as the presence of non-free choice constructs, for example. This underlines the strength of the weighting mechanism as described above, as it now becomes possible to consider all candidate negative events in further analysis tasks, taking into account the confidence measure given to their existence.



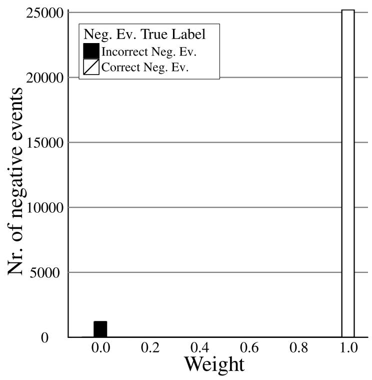
Event log *a5*.



Event log *a6nfc*.

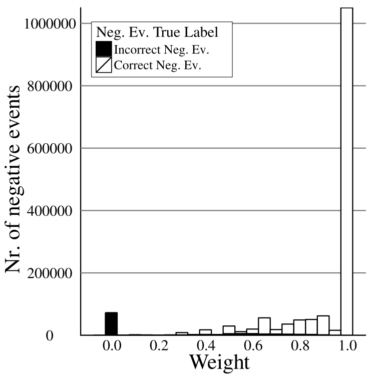


Event log *betaSimplified*.

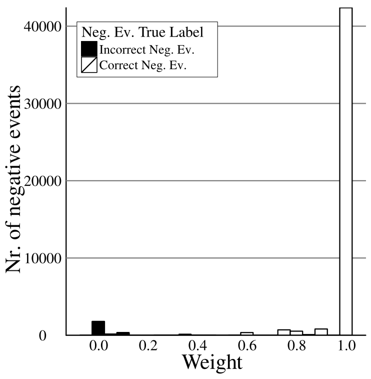


Event log *choice*.

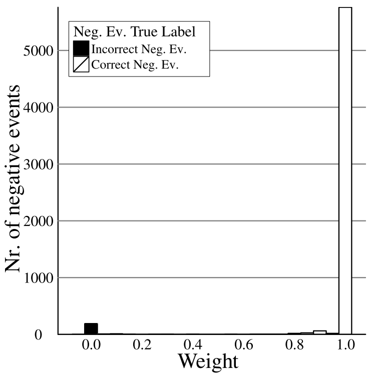
Figure 3.3: Overview of generated artificial negative events with their calculated weights. White colored bars represent correct negative events as indicated by the reference model, whereas black coloring indicates incorrect negative events. The distributions show that, for most candidate negative events, the weighting technique is able to correctly and unambiguously indicate whether the negative event is valid (weight of 1) or invalid (weight of 0).



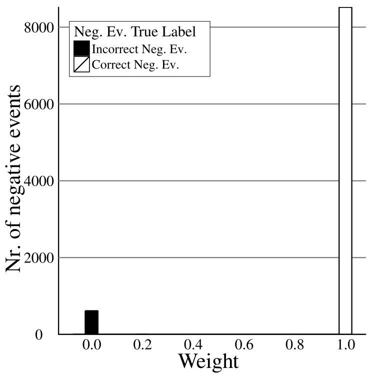
Event log *complex*.



Event log *driversLicenseLoop*.

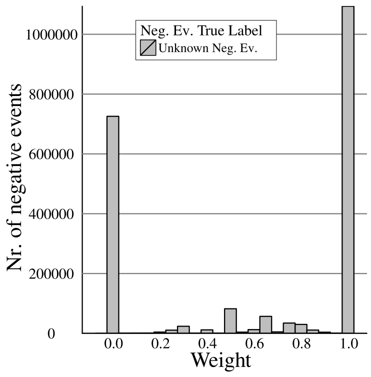


Event log *herbstFig3p4*.

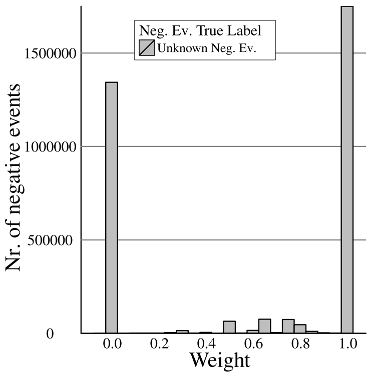


Event log *l2lOptional*.

Figure 3.3 (continued): Overview of generated artificial negative events with their calculated weights.



Event log *incident*.

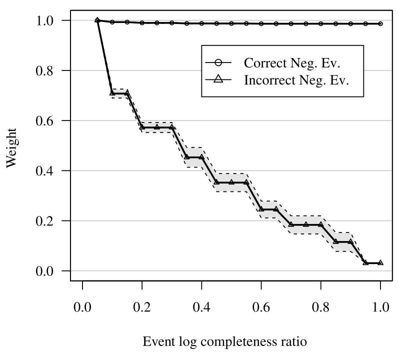


Event log *telecom*.

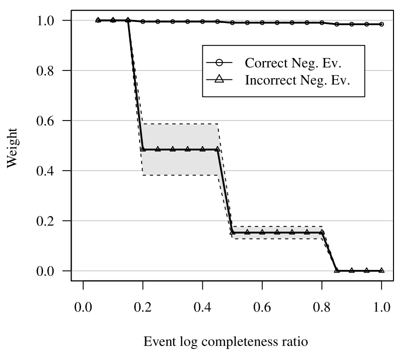
Figure 3.3 (continued): Overview of generated artificial negative events with their calculated weights. For real-life logs *incident* and *telecom*, no reference model is available, so that only the distribution of generated negative event weights are given (colored grey).

3.3.3.2 Robustness

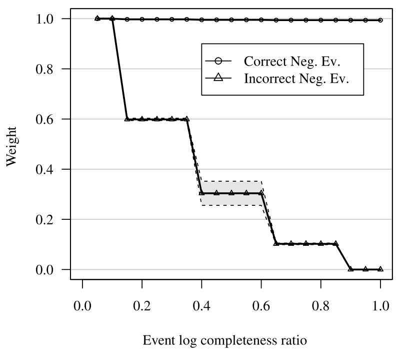
The second validation task investigates how the weighting given to negative events evolves in comparison with the completeness of the given input event log. Figure 3.4 depicts the evolution of negative event weights in comparison with the completeness of the log. An event log generator was developed and used to simulate complete event logs (bounded in the number of loops allowed) from the reference Petri nets (the generation of event logs from a Petri net will be discussed in detail in Chapter 8). Starting from this complete log, distinct traces were randomly removed to obtain smaller sized, less complete logs. More precisely, to generate the artificial negative events, the input log in which negative events are induced is kept constant over all runs, corresponding with the logs listed in Table 3.1, whereas the log used to build the suffix tree is modified for each run and set equal to the differently sized logs. This ensures that the same negative events are generated in each run, allowing to better compare the evolution of their weights. If the differently sized logs themselves would have been used to induce the negative events herein, their varying sizes would impact the average weight for the correct and incorrect negative events, as an increase in log size could create a large additional amount of negative events, making it impossible to track their global weight evolution. This operation was repeated twenty times. The results again support our requirement that correct negative events are generally given a higher weight than incorrect ones, and that this difference further increases as the input event log becomes more complete. For some logs, however, the mean weight obtained for the incorrect negative events remains high when these logs are very incomplete. This behavior is somewhat expected and desired, as it could be argued that the reference model can no longer be accepted as the ideal solution in terms of quality when the input event log gets too small. As such, the aim of assigning a weighting to artificially induced negative events is not to uncover the true underlying reference model behind a log, even when the log is very incomplete (this would be impossible, in fact), but rather to weaken the completeness requirement in a robust and correct manner, so that less complete event logs can still be used to evaluate more complex process models without negatively impacting the quality assessment. Remark that all artificial logs listed in Table 3.1 are sufficiently complete, to ensure that their corresponding reference model remains the optimal solution quality-wise.



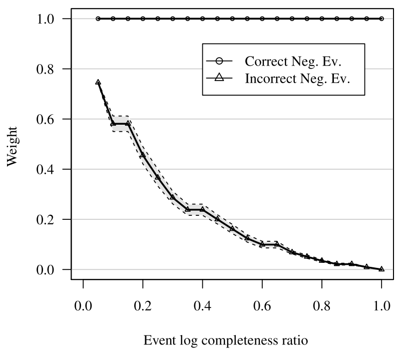
Event log *a5*.



Event log *a6nfc*.

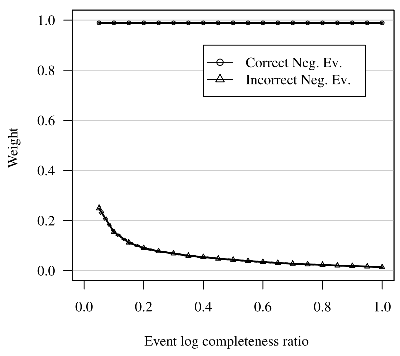


Event log *betaSimplified*.

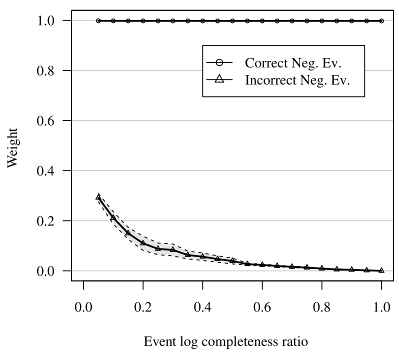


Event log *choice*.

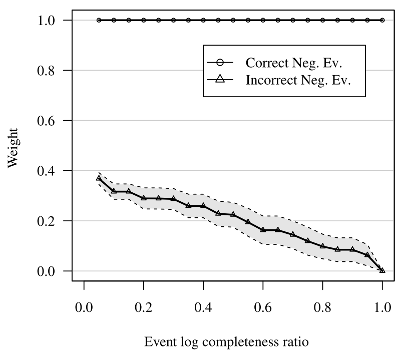
Figure 3.4: Evolution of artificial negative event weights in comparison with event log completeness, averaged over twenty iterations with the 95% confidence interval shown above and below the average. Negative events with low weights are refuted as additional traces are added (i.e. reach a weight of 0) whereas negative events with higher weights remain stable.



Event log *driversLicenseLoop*.



Event log *herbstFig3p4*.



Event log *l2lOptional*.

Figure 3.4 (continued): Evolution of artificial negative event weights in comparison with event log completeness.

3.4 Checking Precision and Generalization

Based on the definition of weighted artificial negative events as explained in the previous section, we now introduce two new metrics, *Weighted Behavioral Precision* (p_B^w) and *Weighted Behavioral Generalization* (g_B^w), to assess the conformance of process models in accordance with a given event log.

To calculate the precision and generalization of a process model in comparison with a given log, all traces $\sigma \in L$ are replayed on the given process model. For each such trace, we calculate values according with the sets of true positive events TP (positive events which could be replayed without error), false positive events FP (negative events which could be replayed and are thus erroneously permitted by the process model), allowed generalizations AG (generalized events which could be replayed without error and confirm the model's ability to generalize) and disallowed generalizations DG (generalized events which could not be replayed by the process model). The concept of allowed and disallowed generalizations requires some further explanation. Consider the trace $\sigma = \langle a, b, d, e \rangle$ from a log L with $A_L = \langle a, b, c, d, e \rangle\}$. We induce artificial negative events, so that $\sigma^- = \langle (b^-, c^-, d^-, e^-), a, (a^-, d^-, e^-), b, (a^-, b^-, c^-, e^-), d, (a^-, b^-, c^-, d^-), e \rangle$ (assume the weight of each negative event equal to 1, i.e. complete confidence). Consider now the positive event $\sigma_2 = b$. The set of negative events preceding this positive event is $NE(\sigma_i) = \{a^-, d^-, e^-\}$. By comparing these events with the full activity alphabet A_L, it is possible to deduce which alternative events should also be permitted by the process model after the execution of a (i.e. before $\sigma_2 = b$), namely $A_L \backslash \{b, a, d, e\} = \{c\}$ (the activity alphabet minus the positive event and its preceding negative events). The process model can then be queried to investigate whether these deduced generalized events are indeed accepted or not. We have assumed the negative events in this example to have a weight of 1; it follows that the strength of a negative event also indicates something about the strength of this candidate event towards generalization. In short: negative events with a weight of 1 are unusable to assess generalization capabilities of a process model, whereas negative events with a weight of 0 (the ones not listed in trace σ^-) are completely unusable to assess precision, but fully valid to determine a model's capability to generalize. Consequently, negative events with a weight between the extrema of 0 and 1, impact both precision and generalization. From now on, we will thus assume that, when artificial negative events are generated in traces, all activities in A_L are inserted before each positive event, excluding the positive event itself, together with their associated weights.

The values for TP, FP, AG and DG are now calculated as follows between an event log L and a process model. Each trace $\sigma \in L$ is replayed on the process model. For every positive event $\sigma_i \in \sigma$, function $NE(\sigma_i)$ is called to induce the set containing all artificial negative events with their weights at this position. Starting from the state reached so far in the trace replay (e.g. the current marking in case of a Petri net), we inspect whether each negative event $n \in NE(\sigma_i)$ could be fired by the process model. If this is indeed the case, the value of FP is incremented with the weight of the negative event $\mathit{Weight}(n)$, and the value of AG is incremented with $1 - \mathit{Weight}(n)$. If the process model is unable to parse the negative event, DG is incremented with $1 - \mathit{Weight}(n)$. Remark that we only inspect the possibility of executing each negative event, without actually firing the corresponding process model element. After evaluating the negative events at the current position towards checking precision and generalization conformance, the positive event itself is parsed. When this event could be fired without error, the value of TP is increased by 1.

Concerning the implementation of trace replay, a heuristic procedure has been developed to evaluate event logs on Petri nets, similar to other approaches described in literature [2, 99]. However, evaluating traces containing negative events requires a modified trace replay procedure, which explores from the marking obtained after firing the last encountered positive event all invisible transition "paths" in order to determine if a negative event can be enabled or not, an aspect which is not taken into account in earlier techniques. Additionally, some words should be devoted to the aspect of force firing. As the force firing of transitions produces additional tokens, it might follow that these tokens subsequently lead to the undesirable enabling of negative events. Adriansyah et al. have proposed a replay technique which avoids force firing altogether by allowing the process model and log to move independently (thus resulting in a model-log alignment). As such, we have added a second replay procedure to establish a best fitting firing sequence based on the alignment technique as described in [108–112], which evaluates positive events based on whether the event under consideration could be successfully aligned with a transition execution in the process model. For negative events, the same evaluation procedure as before is utilized (with invisible path exploration), starting from the marking obtained after the last aligned model-log move. In our heuristic approach, we skip the evaluation of the set of negative events following immediately after a force fired positive event. More details on how to replay an event log with negative events in a Petri net are described in Subsection 3.6.1. Note that our heuristic approach effectively implements a method to calculate a firing sequence $f^{M_0,(P,T,F)}$, as was mentioned in the introductory chapter.

The precision and generalization metrics are now calculated as:

$$p_B^w = \frac{TP}{TP + FP} \qquad\qquad g_B^w = \frac{AG}{AG + DG}$$

Remark that the weight of a negative event (its impact on precision) was defined as $\mathrm{Min}_{\forall \textit{window comparisons}}(\frac{|\textit{window}|-|\textit{matching window}|}{|\textit{window}|})$, i.e. the *minimal unmatching window ratio* over all window comparisons performed. It follows thus, that the impact of a negative event on generalization is equal to:

$$1 - \mathrm{Min}_{\forall \textit{window comparisons}}(\frac{|\textit{window}| - |\textit{matching window}|}{|\textit{window}|})$$

or, also:

$$\mathrm{Max}_{\forall \textit{window comparisons}}(\frac{|\textit{matching window}|}{|\textit{window}|})$$

i.e. the *maximal matching window ratio* found over all window comparisons performed.

We have, thus far, presented a unified technique to calculate precision and generalization based on event traces with weighted artificial negative events. Note finally that fitness can still be calculated as in [2] (i.e. using the true positive and false negative values):

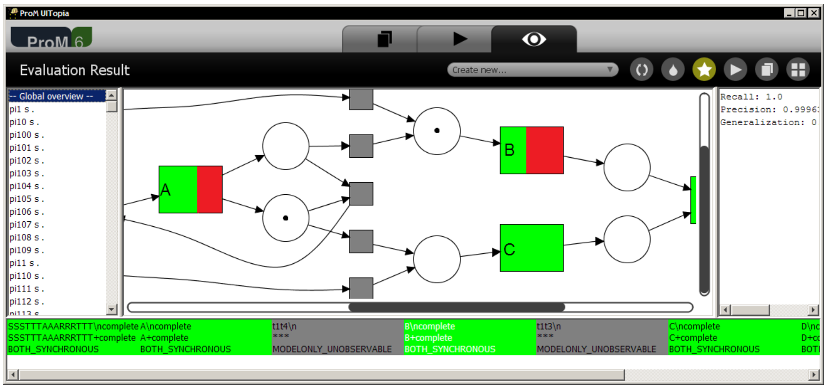
$$r_B = \frac{TP}{TP + FN}$$

With FN the positive events which could not be replayed and are thus erroneously rejected by the process model.

All described techniques and methods are implemented in a series of ProM plugins. Figure 3.5 shows a screen capture illustrating the implementation.

3.5 Experimental Evaluation

This section presents a comparative analysis of our proposed *Weighted Behavioral Precision* (p_B^w) and *Weighted Behavioral Generalization* (g_B^w) metrics in respect to other approaches found in the literature.

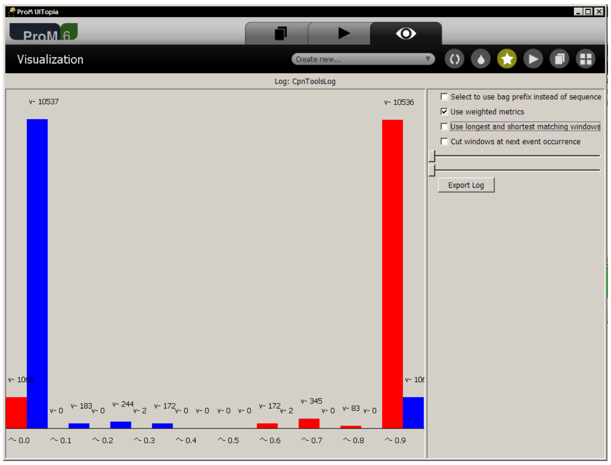


-- Global overview --
pi1 s .
pi10 s .
pi100 s .
pi101 s .
pi102 s .
pi103 s .
pi104 s .
pi105 s .
pi106 s .
pi107 s .
pi108 s .
pi109 s .
pi11 s .
pi110 s .
pi111 s .
pi112 s .

```
Recall: 1.0
Precision: 0.9996
Generalization: 0
```

```
SSSTTTAAARRRTTT\ncomplete A\ncomplete   t1t4\n                  B\ncomplete           t1t3\n                  C\ncomplete        D\nc
SSSTTTAAARRRTTT+complete  A+complete    ***                     B+complete           ***                     C+complete        D+c
BOTH_SYNCHRONOUS          BOTH_SYNCHRONOUS MODELONLY_UNOBSERVABLE BOTH_SYNCHRONOUS   MODELONLY_UNOBSERVABLE  BOTH_SYNCHRONOUS  BOT
```

Figure 3.5: Screen capture implemented ProM plugin. The left panel shows the (distinct) trace variants contained in the given event log. The right panel shows key metrics obtained from the replay/evaluation procedure. The bottom panel allows users to step through events and inspect which transitions were (force) fired accordingly. Finally, the main central area contains the Petri net itself. Transitions are annotated with values corresponding with the number of times the transition occurred as a true/false positive/negative or as an allowed or disallowed generalized event. Transitions and places are also color-coded to allow users to quickly see where and why conformance problems occur.



☐ Select to use bag prefix instead of sequence
☑ Use weighted metrics
☐ Use longest and shortest matching windows
☐ Cut windows at next event occurrence

Figure 3.6: An additional plugin was developed to inspect to weighting distribution of negative and generalized event in a visual manner.

3.5.1 Setup

The following techniques are included in the experimental setup (see Table 3.2 for an overview). For precision, the *Advanced Behavioral Appropriateness* (a'_B) metric is included, as defined by Rozinat et al. [100]. Although this metric provides a theoretically sound method in order to evaluate the precision of a process model, it has been argued that a number of drawbacks exist, among which an exhaustive calculation requirement and an implementation which is only approximate [107]. Second, the *ETC Precision* metric (etc_P) as proposed by Muñoz-Gama and Carmona [105, 106] evaluates precision based on the concept of so-called escaping edges. Adriansyah et al. define a similar precision metric as *ETC Precision* (etc_P) in [109], based on their model-log alignment replay technique, denoted here as *Alignment Based Precision* (p_A). The same authors have also defined *One Align Precision* (a^1_P) and *Best Align Precision* (a_P) metrics [112], which combine the concept of model-log alignments with the metrics defined by Muñoz-Gama and Carmona. Adriansyah et al. also propose a generalization metric in [109] where a Bayesian estimator is applied in order to derive a model's ability to generalize, based on the idea that the likelihood of new, unseen behavior in a certain state depends on the number of times this state was encountered and the number of different decisions (i.e. activities) observed in this state. This technique has been included as *Alignment Based Generalization* (g_A). In [100], an *Advanced Structural Appropriateness* (a'_S) metric is defined to determine whether a process model overfits an event log. However, this metric only takes into account certain structural properties of a process model (redundant invisible transitions and alternative duplicate transitions), so that it is not fully able to evaluate generalization. More precisely, it is argued that this metric better fits with the fourth quality dimension—simplicity, as redundant and duplicate transitions mainly hinder a model's ability to be well and easily understood than its ability to parse event traces. As such, we do not include this entry in our setup. Finally, we compare our approach with a related technique described by De Weerdt et al. [107] and Goedertier et al. [2], where the notion of precision (*Behavioral Precision,* p_B) based on negative events was initially put forward. Since our technique extends both the generation procedure of negative events and the replay procedure applied, we expect to gain a performance increase. This earlier approach does not define a generalization metric. Other, less related, precision and generalization metrics exist, which either target a process modeling representation other than Petri nets or are unimplemented in ProM, so that the set of evaluated conformance metrics is limited to the entries described above.

Table 3.2: Conformance checking measures included in the experimental setup.

Name	Symbol	Author(s)	Quality Dimension
Behavioral Recall	r_B	vanden Broucke et al. [101]	Recall
Advanced Behavioral Appropriateness	$a_B^{'}$	Rozinat and van der Aalst [100]	Precision
ETC Precision	etc_p	Muñoz-Gama et al. [105, 106]	Precision
Behavioral Precision	p_B	De Weerdt et al. [107]	Precision
Alignment Based Precision	p_A	Adriansyah et al. [109]	Precision
One Align Precision	a_p^1	Adriansyah et al. [112]	Precision
Best Align Precision	a_p	Adriansyah et al. [112]	Precision
Weighted Behavioral Precision	p_B^w	vanden Broucke et al. [101]	Precision
Alignment Based Probabilistic Generalization	g_A	Adriansyah et al. [109]	Generalization
Weighted Behavioral Generalization	g_B^w	vanden Broucke et al. [101]	Generalization

The included metrics are compared with our new conformance checking approach, i.e. *Weighted Behavioral Precision* (p_B^w) and *Weighted Behavioral Generalization* (g_B^w), using both our own heuristic trace replay method and the replay method based on model-log alignments. For reasons of completeness and to confirm the capability of the developed replay techniques to correctly parse traces (based on the known reference and example models), we report *Behavioral Recall* (r_B) in the results as well (also using both replay methods).

3.5.2 Fixed Log Sizes

In this comparative analysis experiment, the event log test set introduced in Section 3.3 will be utilized. Remark that most event logs stem from a-priori known reference models and will be evaluated on the same model. As such, we expect well-performing conformance checking techniques to reach high precision and generalization scores for these combinations. For the real-life logs, no reference model is known, so that we will evaluate these logs on discovered Petri net models (using the Heuristics Miner [44] discovery algorithm). Finally, we also incorporate the five models from Figure 3.1 in our experimental setup (event log: *exampleLog*), as this allows us to evaluate which conformance checking technique is best able to punish Petri net models of extremely low quality (low precision for *connectedModel* and *flowerModel,* and low generalization for *singleModel* and *stackedModel*).

Table 3.3 shows the results of our experiment for the evaluated metrics on all event log-Petri net pairs. For each result, scores are reported as a number between 0 (worst) and 1 (best). Run times are given between parentheses as seconds. Calculations were limited to 24 hours of computational time. Runs exceeding this amount of time or runs which resulted in a crash or error are left out("–"). All experiments were executed on a workstation with 2 processors (2.53Ghz; 4 cores per processor) and 64GB of memory. The Java heap size for each individual experiment was set to 4GB. Stack size was left default, as no single stack overflow error occurred during the experimentation.

Based on the results listed in Table 3.3, the following observations can be made. First, many conformance checking techniques have difficulties dealing with large or complex event logs. For many of the real-life logs, only the artificial negative event-based conformance checking techniques and *ETC Precision* ($\mathrm{etc_P}$) were able to always find a result, the others encountering out-of-time or out-of-memory errors; note that the *hospital* log forms a noteworthy exception—as the artificial negative event-based conformance checking techniques were not able to compute a result in time. Next, focusing on precision, it is found that many metrics unduly punish precision errors due to input logs being less complete. For all artificial cases, our *Weighted Behavioral Precision* metric ($\mathrm{p_B^w}$) is able to find a precision value close to 1. For the real-life logs, the existing metrics return a lower value than the results obtained by *Weighted Behavioral Precision* ($\mathrm{p_B^w}$). Although the true precision of these mined models is unknown, the almost-0 score obtained by *ETC Precision* ($\mathrm{etc_P}$) appears to be overly pessimistic. Concerning the *exampleLog* models, we observe that for *connectedModel* and *flowerModel*, the *Weighted Behavioral Precision* metric ($\mathrm{p_B^w}$) correctly punishes the many precision errors found in these models. Note that some metrics have issues when traversing the possible invisible paths in *connectedModel*, with *Behavioral Precision* ($\mathrm{p_B}$) being unable to detect any imprecisions at all. Concerning generalization, the *Alignment Based Generalization* metric ($\mathrm{g_A}$) returns unexpectedly high values for *stackedModel* and *singleModel* (the score of 0 for *driversLicense* is also unexpected and perhaps due to a bug). The *Weighted Behavioral Generalization* metric ($\mathrm{g_B^w}$) is able to punish each generalization issue, and does so in a more strict manner. Therefore, for real-life applications, it remains recommended to first focus on model fitness and precision, followed by generalization with lower priority. A final remark should be made concerning the differences between the two implemented replay techniques (heuristic and alignment based) when using the *Weighted Behavioral Precision* and *Generalization* metrics ($\mathrm{p_B^w}$ and $\mathrm{g_B^w}$). For many logs, we cannot observe a significant difference in values between the two techniques, although the run times are higher when using the alignment based replay procedure. Some interesting differences are observed for the *telecom*, *incident* and the *connectedModel* and *singleModel* logs, where the alignment based replay procedure is less punishing with regards to precision.

Table 3.3: Obtained values and run times (in seconds) for the evaluated recall metrics.

Event Log	Petri Net	Recall Metric	
		r_B (heuristic)	r_B (alignment)
a10skip	*reference model*	1.00 (0.31s)	1.00 (0.61s)
a12	*reference model*	1.00 (0.30s)	1.00 (0.55s)
a5	*reference model*	1.00 (0.36s)	1.00 (0.58s)
a6nfc	*reference model*	1.00 (0.17s)	1.00 (0.39s)
a7	*reference model*	1.00 (0.34s)	1.00 (0.55s)
a8	*reference model*	1.00 (0.22s)	1.00 (0.37s)
betaSimplified	*reference model*	1.00 (0.31s)	1.00 (0.47s)
choice	*reference model*	1.00 (0.44s)	1.00 (0.70s)
complex	*reference model*	1.00 (3.20s)	1.00 (10.34s)
driversLicense	*reference model*	1.00 (0.09s)	1.00 (0.17s)
driversLicenseLoop	*reference model*	1.00 (1.02s)	1.00 (1.65s)
herbstFig3p4	*reference model*	1.00 (0.50s)	1.00 (0.98s)
herbstFig5p19	*reference model*	1.00 (0.33s)	1.00 (0.44s)
herbstFig6p18	*reference model*	1.00 (0.95s)	1.00 (4.11s)
herbstFig6p31	*reference model*	1.00 (0.25s)	1.00 (0.56s)
herbstFig6p36	*reference model*	1.00 (0.34s)	1.00 (0.64s)
herbstFig6p38	*reference model*	1.00 (0.33s)	1.00 (0.51s)
herbstFig6p41	*reference model*	1.00 (0.47s)	1.00 (0.89s)
l2l	*reference model*	1.00 (0.30s)	1.00 (0.48s)
l2lOptional	*reference model*	1.00 (0.31s)	1.00 (0.47s)
l2lSkip	*reference model*	1.00 (0.30s)	1.00 (0.50s)
exampleLog	*perfectModel*	1.00 (0.64s)	1.00 (0.69s)
exampleLog	*connectedModel*	1.00 (0.83s)	1.00 (0.95s)
exampleLog	*flowerModel*	1.00 (0.37s)	1.00 (0.70s)
exampleLog	*singleModel*	0.38 (0.45s)	0.45 (0.64s)
exampleLog	*stackedModel*	0.84 (0.86s)	1.00 (0.97s)
hospital (real-life)	*mined model*	0.92 (5460.67s)	–
incident (real-life)	*mined model*	0.79 (13.56s)	0.41 (9374.49s)
telecom (real-life)	*mined model*	0.70 (15.70s)	0.39 (16523.82s)
ticketing (real-life)	*mined model*	1.00 (503.24s)	1.00 (701.53s)

Table 3.3 (continued): Obtained values and run times (in seconds) for the evaluated precision metrics.

Event Log	Petri Net	Precision Metrics						Weighted Behavioral Precision	
		$a_B^{\prime}$	etc_P	p_A	a_P^1	a_P	p_B	p_B^w (heuristic)	p_B^w (alignment)
a10skip	*reference model*	1.00 (0.33s)	0.91 (0.25s)	0.97 (0.42s)	1.00 (0.50s)	0.92 (5.60s)	1.00 (1.81s)	1.00 (0.53s)	1.00 (0.61s)
a12	*reference model*	1.00 (0.35s)	1.00 (0.23s)	1.00 (0.41s)	1.00 (0.42s)	0.93 (4.38s)	1.00 (3.26s)	1.00 (0.45s)	1.00 (0.66s)
a5	*reference model*	0.47 (0.38s)	0.29 (0.22s)	1.00 (0.53s)	0.93 (0.45s)	0.70 (13.50s)	1.00 (0.70s)	1.00 (0.47s)	1.00 (0.70s)
a6nfc	*reference model*	0.69 (0.35s)	0.51 (0.22s)	1.00 (0.34s)	0.79 (0.41s)	0.78 (2.31s)	0.94 (0.53s)	1.00 (0.30s)	1.00 (0.49s)
a7	*reference model*	1.00 (0.33s)	1.00 (0.23s)	1.00 (0.39s)	1.00 (0.48s)	0.77 (8.84s)	1.00 (0.72s)	1.00 (0.44s)	1.00 (0.72s)
a8	*reference model*	1.00 (0.30s)	1.00 (0.22s)	1.00 (0.30s)	1.00 (0.33s)	0.80 (1.89s)	1.00 (0.74s)	1.00 (0.28s)	1.00 (0.45s)
betaSimplified	*reference model*	1.00 (0.47s)	0.77 (0.27s)	0.98 (0.42s)	0.94 (0.49s)	0.87 (4.55s)	1.00 (7.91s)	1.00 (0.45s)	1.00 (0.62s)
choice	*reference model*	1.00 (0.36s)	0.57 (0.27s)	1.00 (0.60s)	1.00 (0.63s)	0.89 (19.76s)	1.00 (1.58s)	1.00 (0.61s)	1.00 (0.91s)
complex	*reference model*	–	0.55 (1.52s)	0.75 (5.33s)	0.65 (3.08s)	–	0.73 (1963.23s)	0.73 (18.54s)	0.74 (25.58s)
driversLicense	*reference model*	1.00 (0.11s)	0.63 (0.03s)	1.00 (0.13s)	0.90 (0.11s)	0.80 (1.30s)	1.00 (0.13s)	1.00 (0.17s)	1.00 (0.36s)
driversLicenseLoop	*reference model*	0.91 (0.53s)	0.90 (0.41s)	0.95 (1.06s)	0.90 (0.84s)	–	0.87 (10.02s)	0.97 (1.78s)	0.97 (3.78s)
herbstFig3p4	*reference model*	0.70 (0.28s)	0.99 (0.22s)	0.98 (0.44s)	0.97 (0.34s)	0.92 (77.55s)	0.96 (3.50s)	0.99 (1.12s)	0.99 (2.23s)
herbstFig5p19	*reference model*	1.00 (0.48s)	0.88 (0.25s)	0.96 (0.42s)	1.00 (0.34s)	0.89 (5.31s)	1.00 (0.70s)	1.00 (0.47s)	1.00 (0.66s)
herbstFig6p18	*reference model*	0.59 (0.55s)	0.40 (0.34s)	0.73 (2.28s)	0.93 (1.80s)	0.88 (1239.92s)	0.91 (7.23s)	0.97 (1.95s)	0.97 (3.79s)
herbstFig6p31	*reference model*	1.00 (0.41s)	1.00 (0.25s)	1.00 (0.34s)	1.00 (0.38s)	0.78 (1.25s)	1.00 (0.72s)	1.00 (0.33s)	1.00 (0.53s)
herbstFig6p36	*reference model*	1.00 (0.53s)	0.75 (0.36s)	1.00 (0.58s)	0.92 (0.84s)	0.92 (3.01s)	1.00 (3.31s)	1.00 (0.55s)	1.00 (0.59s)
herbstFig6p38	*reference model*	1.00 (0.52s)	0.90 (0.27s)	1.00 (0.53s)	0.95 (0.44s)	0.89 (5.31s)	1.00 (0.81s)	1.00 (0.42s)	1.00 (0.69s)
herbstFig6p41	*reference model*	1.00 (0.47s)	1.00 (0.39s)	1.00 (0.55s)	1.00 (0.58s)	0.92 (16.51s)	1.00 (11.96s)	1.00 (0.69s)	1.00 (0.95s)
l2l	*reference model*	0.56 (0.28s)	1.00 (0.17s)	1.00 (0.33s)	1.00 (0.41s)	0.79 (8.15s)	1.00 (0.84s)	1.00 (0.39s)	1.00 (0.59s)
l2lOptional	*reference model*	0.65 (0.31s)	0.40 (0.19s)	1.00 (0.37s)	1.00 (0.31s)	0.87 (7.04s)	1.00 (0.72s)	1.00 (0.42s)	1.00 (0.66s)
l2lSkip	*reference model*	0.42 (0.28s)	0.85 (0.22s)	0.93 (0.36s)	1.00 (0.39s)	0.83 (6.37s)	1.00 (0.97s)	1.00 (0.42s)	1.00 (0.61s)
exampleLog	*perfectModel*	1.00 (1.00s)	1.00 (0.36s)	1.00 (0.64s)	1.00 (0.52s)	0.78 (15.33s)	1.00 (2.91s)	1.00 (0.86s)	1.00 (1.05s)
exampleLog	*connectedModel*	–	0.06 (0.36s)	0.26 (1.81s)	0.26 (1.08s)	0.27 (35.93s)	1.00 (3.02s)	0.13 (1.50s)	0.13 (1.84s)
exampleLog	*flowerModel*	–	0.16 (0.42s)	0.26 (0.64s)	0.16 (0.48s)	0.26 (3.67s)	0.12 (2.66s)	0.12 (0.72s)	0.12 (0.98s)
exampleLog	*singleModel*	1.00 (0.70s)	1.00 (0.34s)	1.00 (0.50s)	1.00 (0.59s)	1.00 (1.06s)	0.41 (2.41s)	0.71 (0.63s)	1.00 (0.87s)
exampleLog	*stackedModel*	1.00 (14.31s)	0.12 (0.37s)	1.00 (1.11s)	1.00 (0.86s)	0.71 (123.51s)	0.71 (3.23s)	0.67 (1.19s)	1.00 (1.59s)
hospital (real-life)	*mined model*	–	0.03 (4.68s)	0.13 (1829.74s)	0.03 (1234.47s)	0.04 (873.27s)	–	–	–
incident (real-life)	*mined model*	–	0.05 (1.65s)	–	–	–	0.68 (88.39s)	0.41 (140.83s)	0.94 (5885.06s)
telecom (real-life)	*mined model*	–	0.10 (1.53s)	–	0.47 (17439.21s)	–	0.47 (86385.41s)	0.45 (566.04s)	0.88 (18236.50s)
ticketing (real-life)	*mined model*	–	0.13 (128.31s)	0.81 (15746.07s)	0.99 (15374.66s)	–	1.00 (44834.75s)	0.97 (455.13s)	0.97 (693.85s)

Table 3.3 (continued): Obtained values and run times (in seconds) for the evaluated generalization metrics.

Event Log	Petri Net	Generalization Metric	Weighted Behavioral Generalization	
		g_A	g_B^w (heuristic)	g_B^w (alignment)
a10skip	*reference model*	1.00 (0.48s)	0.91 (0.55s)	0.91 (0.80s)
a12	*reference model*	1.00 (0.39s)	0.83 (0.48s)	0.83 (0.69s)
a5	*reference model*	1.00 (0.55s)	0.88 (0.56s)	0.88 (0.99s)
a6nfc	*reference model*	1.00 (0.47s)	0.64 (0.33s)	0.64 (0.47s)
a7	*reference model*	1.00 (0.41s)	0.70 (0.47s)	0.70 (0.74s)
a8	*reference model*	1.00 (0.30s)	0.94 (0.28s)	0.94 (0.48s)
betaSimplified	*reference model*	1.00 (0.48s)	0.70 (0.50s)	0.70 (0.75s)
choice	*reference model*	1.00 (0.63s)	1.00 (0.81s)	1.00 (1.24s)
complex	*reference model*	1.00 (5.26s)	0.51 (17.94s)	0.51 (22.37s)
driversLicense	*reference model*	0.00 (0.09s)	0.80 (0.14s)	0.80 (0.28s)
driversLicenseLoop	*reference model*	1.00 (1.22s)	0.83 (1.45s)	0.83 (2.96s)
herbstFig3p4	*reference model*	1.00 (0.41s)	0.94 (0.97s)	0.94 (1.69s)
herbstFig5p19	*reference model*	1.00 (0.38s)	0.77 (0.41s)	0.77 (0.81s)
herbstFig6p18	*reference model*	1.00 (2.62s)	1.00 (2.29s)	1.00 (3.87s)
herbstFig6p31	*reference model*	1.00 (0.45s)	0.57 (0.42s)	0.57 (0.55s)
herbstFig6p36	*reference model*	1.00 (0.53s)	0.57 (0.53s)	0.57 (0.87s)
herbstFig6p38	*reference model*	1.00 (0.58s)	0.50 (0.48s)	0.50 (0.66s)
herbstFig6p41	*reference model*	1.00 (0.56s)	0.73 (0.80s)	0.73 (1.00s)
l2l	*reference model*	1.00 (0.37s)	1.00 (0.44s)	1.00 (0.67s)
l2lOptional	*reference model*	1.00 (0.45s)	1.00 (0.45s)	1.00 (0.70s)
l2lSkip	*reference model*	1.00 (0.41s)	1.00 (0.47s)	1.00 (0.64s)
exampleLog	*perfectModel*	1.00 (0.58s)	0.84 (0.62s)	0.84 (0.94s)
exampleLog	*connectedModel*	1.00 (1.78s)	1.00 (1.42s)	1.00 (1.70s)
exampleLog	*flowerModel*	1.00 (0.55s)	1.00 (0.62s)	1.00 (0.89s)
exampleLog	*singleModel*	1.00 (0.47s)	0.00 (0.59s)	0.00 (0.84s)
exampleLog	*stackedModel*	0.98 (0.94s)	0.33 (1.22s)	0.00 (1.40s)
hospital (real-life)	*mined model*	0.29 (1853.34s)	–	–
incident (real-life)	*mined model*	–	0.52 (181.39s)	0.54 (5709.55s)
telecom (real-life)	*mined model*	–	0.41 (557.31s)	0.46 (18215.54s)
ticketing (real-life)	*mined model*	1.00 (20957.02s)	0.81 (429.81s)	0.81 (672.68s)

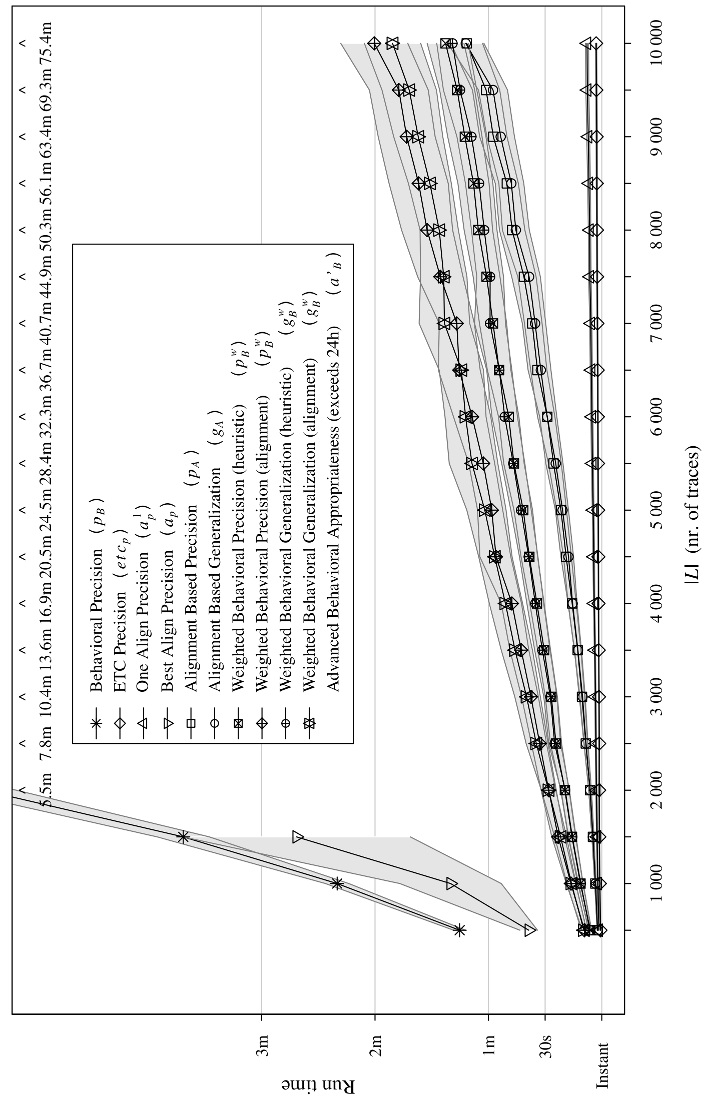


Figure 3.7: Run times (averaged over twenty iterations) of evaluated conformance checking techniques over various log sizes. The grey bands indicate the 95% confidence intervals below and above the means.

3.5.3 Varying Log Sizes

This section discusses the scalability (in terms of run time) and stability (in terms of obtained values) of the evaluated metrics under varying sizes of event logs (and thus completeness). To do so, the process model of the *complex* log was used as a basis from which twenty sets of differently sized event logs (ranging between 500 and 10000 traces) were generated—thus totaling 400 event logs. We then evaluate each log on the reference model using the metrics discussed above.

Regarding scalability, it is investigated whether the run time performance of a metric depends heavily on the size of the evaluated log. Figure 3.7 provides an overview of the required run times for each evaluated technique over various log sizes. The run times of most techniques remain within acceptable bounds. For the *Advanced Behavioral Appropriateness* metric (a_B'), however, calculations always exceeded the allotted 24 hours. This is due to the extensive process model state space exploration which is performed by these metrics. Also, calculating *Best Align Precision* (a_p) requires exponentially more time as the log size is increased. In [112], the authors indeed acknowledge the long calculation time required towards calculating this metric. In addition, for logs containing 2 000 or more traces, the metric was not able to obtain any result due to out-of-memory errors. Concerning the calculation of the artificial negative event-based metrics (*Behavioral Precision* p_B, *Weighted Behavioral Precision* p_B^w and *Weighted Behavioral Generalization* g_B^w), it is important to note that Figure 3.7 includes the time taken to induce the negative events in the reported run time (note the exponential time increase for *Behavioral Precision*). Figure 3.8 shows the run time required for the generation of all artificial negative events separately and illustrates the scalability benefit gained by our implementation using suffix trees to induce the weighted artificial negative events and shows linear complexity in terms of log size, as was discussed above.

Concerning stability, we evaluate whether the obtained precision and generalization values are sensitive to various levels of log completeness, as well as investigate the actual score values themselves. Figure 3.9 depicts an overview of the retrieved metric values. The following remarks can be made based on the results. First, the *Weighted Behavioral Precision* metric (p_B^w) improves slightly upon the non-weighted *Behavioral Precision* (p_B) and *Alignment Based Precision* (p_A) metrics and that it highly exceeds the *ETC Precision* (etc_p) and *One Align Precision* (a_p^1) metrics (recall that, since we are dealing with simulated event logs from a reference model, we expect precision to be high as the evaluated log reaches a

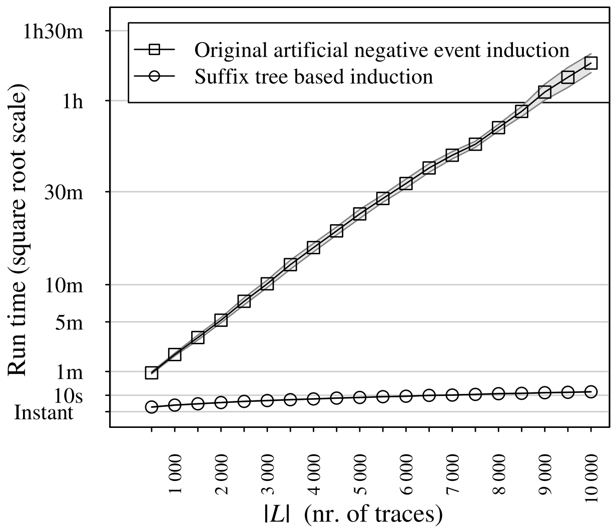


Figure 3.8: Run times (averaged over twenty iterations) of (weighted) artificial negative event procedure over various log sizes, illustrating the speed benefit obtained by our technique compared to the original artificial negative induction method. The grey bands indicate the 95% confidence intervals below and above the means.

sufficient size). On the other hand, the *Best Align Precision* metric (a_p) outperforms our approach, although this metric comes with high run times and is less able to deal with larger event logs. We were not able to confirm the statement made in [112] regarding the similarity in results between *One Align Precision* (a_p^1) and *Best Align Precision* (a_p), since there exists a wide gap between the results for these two metrics. Finally, although the *Advanced Behavioral Appropriateness* metric (a_B') results in the highest precision value, the result is only approximate (the state space exploration phase was canceled after 24h). Moreover, this metric only investigates basic structural relations between model and log activities, so that this metric as well presents a tendency to be overly optimistic. Concerning generalization, the *Weighted Behavioral Generalization* metric (g_B^w) applies the concept of weighted artificial negative events towards checking generalization, which takes into account full behavioral properties found in both process model and event log. Although the metric remains very stable over various log sizes (the slight drop at the beginning is due to the fact that the activity alphabet is incomplete for the very small logs), it is noted that this metric estimates generalization in a very strict manner, especially when compared with the *Alignment Based Generalization* metric (a_g). However, as seen in Subsection 3.5.2, this probabilistic metric also returns very high values for the experiment models with a very low generalization capability, i.e. *singleModel* and *stackedModel*, whereas *Weighted Behavioral Generalization* (g_B^w) does punish these models.

3.6 Discussed Topics

Thus far, we presented a unified technique to calculate precision and generalization based on event traces with weighted artificial negative events. In this section, some additional topics of interest are discussed, among which: more details on the problem of trace replay for traces containing both positive and negative events, additional (optional) configuration parameters which can be modified by expert end-users, a discussion on alternative (i.e. non-sequence window-based) artificial negative event generation strategies, and a discussion on the impact of noise on the generation of artificial negative events.

3.6.1 Trace Replay with Positive and Negative Events

Above, we have introduced a heuristic approach to replay traces containing positive and negative events, so as to evaluate a process model in terms of its recall, precision, and generalization abilities through the construction of a confusion matrix (true/false positive/negative events and allowed and disallowed generalized events). We have stated that our heuristic approach also offers a first method to calculate a firing sequence $f^{M_0,(P,T,F)}$ for a given (positive) log trace, i.e. a series of transitions which can be mapped ot this positive trace. To iterate: a firing sequence $f^{M_0,(P,T,F)}$ is a finite sequence of transitions $\langle t_1, t_2, \ldots, t_n \rangle \in T$, with M_0 the initial marking and M_i ($0 < i \leqslant n$) the resulting marking after (force) firing t_1 up to and including t_i sequentially. A trace $\sigma \in L$ can be replayed by the Petri net (P, T, F) if a firing sequence $f^{M_0,(P,T,F)}$ can be found such that $\sigma_i = \mu(f_i^{M_0,(P,T,F)*})$ with $0 < i \leqslant |\sigma|$ and $f^{M_0,(P,T,F)*} = f^{M_0,(P,T,F)} \setminus \{t \in f^{M_0,(P,T,F)} | \mu(t) = a_i\}$. Note that it is possible that multiple firing sequences can be found to replay a trace. When a firing sequence can be found for a trace so that all transitions can be fired sequentially starting from initial marking M_0 without one of them being forced to fire, it is said that the trace fits in the Petri net. Replay is thus closely related to the first model quality dimension: fitness.

In this section, we offer some additional insights on how exactly a firing sequence can be constructed through a Petri net for a given trace. More specifically, we will look at:

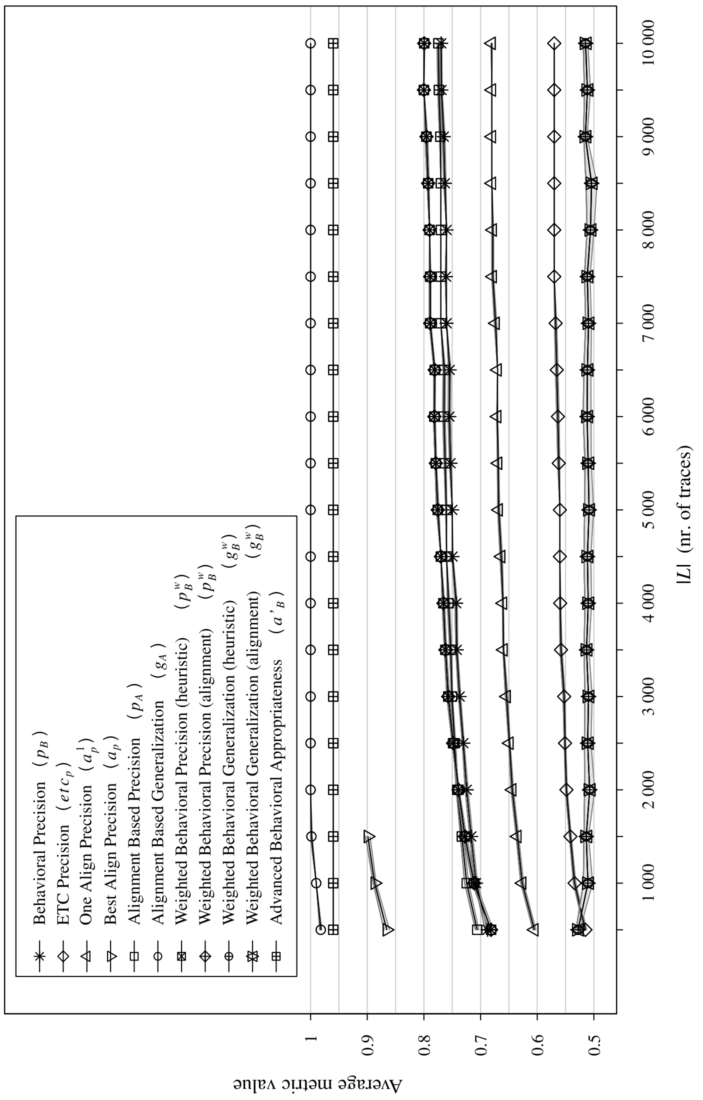


Figure 3.9: Results of evaluated conformance checking techniques (averaged over twenty iterations) over various log sizes. The grey bands (narrow) indicate the 95% confidence intervals below and above the means.

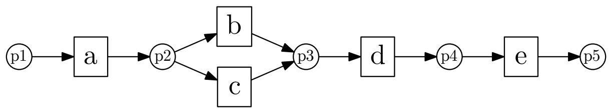


Figure 3.10: A simple process model used as an illustration for trace replay.

- *Replay strategies*: to construct a firing sequence $f^{M_0,(P,T,F)}$ for a given trace $\sigma \in L$. Note that, for the construction of a firing sequence, only the positive events are needed, as negative events can be evaluated separately once a firing sequence is constructed.
- *Evaluation schemes*: determining the manner how negative events $NE(\sigma_i)$ for $\sigma \in L$ are evaluated on a constructed firing sequence $f^{M_0,(P,T,F)}$.

3.6.1.1 Introduction

Consider the trace: $\sigma = \langle a, b, d, e \rangle$ from a log L with $A_L = \{a, b, c, d, e\}$. This trace can be replayed on the process model in Figure 3.10 in the following manner (constructing the transition-activity mapping μ is trivial in this case: each transition is mapped to the activity in A_L with the same name). The initial marking is $M_0 = \{p1\}$. We can easily determine which transition to fire to execute event a (there is only one candidate), so that $M_1 = \{p2\}$. The next markings (after firing b, d and e) are $M_2 = \{p3\}$, $M_3 = \{p4\}$ and final marking $M_4 = \{p5\}$. We thus simply obtain the following fitting firing sequence:

$f^{M_0,(P,T,F)}$	a	b	d	e
σ	a	b	d	e

.

Suppose now that we have a trace $\tau = \langle a, d, c, e \rangle$. After firing a, the marking is once more $M_1 = \{p2\}$. Since there is now only one candidate-transition to fire activity d, and this transition is not enabled (there is no token at $p3$ for M_1), the transition is force fired and $M_2 = \{p2, p4\}$. After subsequently executing c and e, the final marking equals $M_4 = \{p3, p5\}$. Note the remaining token at place $p3$. We thus obtain the following firing sequence:

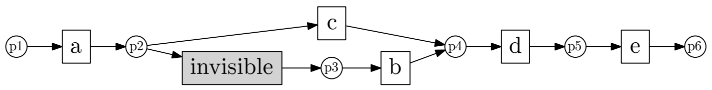


Figure 3.11: A simple process model used as an illustration for trace replay (invisible activity added).

$$\begin{array}{r|cccc} f^{M_0,(P,T,F)} & a & d* & c & e \\ \hline \tau & a & d & c & e \end{array}.$$

Note that this firing sequence is not completely fitting, as a transition (d) was force fired in order to execute the trace τ.

The problem of trace replay in the context of process mining has been extensively described in the literature [99]. Generally speaking, two essential root causes exist which make replaying traces on Petri nets hard. First, in the presence of invisible transitions, it is non-trivial to determine whether there exists a suitable firing sequence of invisible transitions so that the correct activities become enabled for the Petri net to optimally replay a trace (a simplified case is shown in Figure 3.11: before firing b, the invisible transition *invisible* should be fired first). In extreme cases, it is possible that an invisible transition should be fired even though its execution is not necessary to fire the first-next activity at hand. This is often described as "non-lazy" firing of invisible transitions, for example, in Figure 3.12, the invisible transition *invisible* should be fired in order to enable b, even although this is not required to fire a. Second, in the presence of multiple enabled duplicate transitions, it is hard to determine the optimal firing, as firing one duplicate activity affects the ability of the Petri net to replay the remaining events in the given trace (see Figure 3.13 for a simplified example where transition b is duplicated and the decision of which transitions has to be fired depends on the further progression of the trace at hand). Again, in extreme cases, it is theoretically possible that the decision to fire a *non-enabled* transition should be preferred above the firing of an enabled one, when this force fire results in an overall better fitness result for the given trace.

Following from the description of the two main issues present in trace replay as described above (invisible and duplicate transitions), it becomes clear that also the evaluation of negative events should be modified such that possible sequences of invisible transitions which would lead to the enabling of the negative

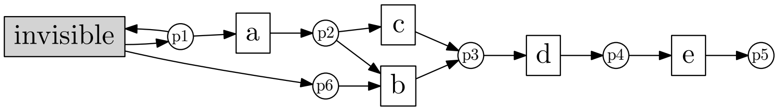


Figure 3.12: A simple process model used as an illustration for trace replay (another example with invisible activity).

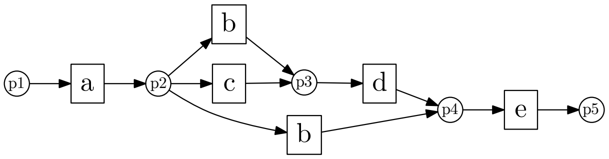


Figure 3.13: A simple process model used as an illustration for trace replay (duplicate activity).

event under consideration should be checked as well. Consider for example the partial trace $\langle a, \{b^-\}\rangle$ to be evaluated on the model in Figure 3.11. Although it would seem that, after firing a, b is not enabled (marking $M_1 = \{p2\}$, it is clear that an invisible path exists (consisting of a single transition: *invisible*) which does enable b. Remark that this approach effectively evaluates negative events as they are encountered while stepping through a given trace, i.e. as this evaluation occurs starting from the marking obtained after the execution of the last transition (corresponding with a positive event), we classify this approach as being "*state-bound*".

Due to this state-boundedness, however, the technique described above is not able to deal with cases where the decision to fire an invisible transition (or a series of invisible transitions) leading to the enabling of a negative event under consideration should be made before encountering the last seen positive event (i.e. the non-lazy choices). To deal with this issue, it no longer suffices to start from the marking obtained after firing the last seen positive event to evaluate a negative event. Instead, a new trace should be constructed, consisting of the positive event prefix up until the negative event (i.e. all positive events before the negative event at hand), with the negative event concatenated at the end. The process model is then queried again (starting from the initial marking) to see whether it is able to replay this constructed ("negative") trace. This "*state-free*" approach evaluates each negative event independently from the decision chain of transitions made before the occurrence of the negative event, but also requires more computational time, as each evaluated negative event now entails the construction and subsequent replay of new traces.

Another argument can be made to this regard which also illustrates the difference between the two replay approaches outlined above, i.e. evaluating negative events as they are encountered by stepping through the trace (state-bound) versus evaluating negative events on their own and independently from the currently-reached marking by constructing a new trace (state-free). One could argue that the former state-bound approach is more in line with the way a human end-user or running system would execute a trace, where it is deemed unlikely that one wishes to navigate or backtrack through a labyrinth of transitions in order to find the best firing sequence. From this *execution* perspective, it thus makes sense that we evaluate negative events from the currently-reached marking and that we commit ourselves to earlier made decisions (meaning the sequence of fired transitions and its resulting state). The latter described state-free method on the other hand, where a new trace is constructed to check each negative event independently does not correspond with the way end users execute processes, but rather evaluate a given process model based purely on its ability to reach or not reach certain states, i.e. from an *evaluation* perspective.

Let us now formalize the different replay strategies and negative event evaluation schemes as outlined above in more detail.

3.6.1.2 Replay Strategies

This section lists the various strategies which can be applied to construct a firing sequence $f^{M_0,(P,T,F)}$ given a trace σ. Note that, for the construction of a firing sequence, only the positive events are needed, as negative events can be evaluated separately once a firing sequence is constructed.

Complete Replay: Full Exploration This replay strategy considers all possible transition sequences which can be mapped to the trace σ. Algorithm 3.3 provides a concise formalized overview of the workings of this replay strategy. For each event encountered in the trace, a list of candidate transitions is constructed, consisting of all enabled invisible activities and all transitions mapped to the event at hand, either enabled or not. Next, the search is recursively branched on all candidate transitions and further explored.

Note that some further optimizations can be added which are not explicitly shown in Algorithm 3.3:

- First, the list of candidate transitions can be ordered so that enabled, mapped transitions are considered first, followed by invisible transitions and finally, disabled mapped transitions which will be force fired. This is useful in cases where one only wishes to obtain one single firing sequence, preferably one containing a minimal amount of force fired transitions.
- Next, in cases where one wishes to obtain the one single firing sequence with the absolute lowest number of force fired transitions, a complete exploration is still required, but the current best firing sequence can then be used to cut off the search in cases where the firing sequence f so far contains more force fired transitions than the current best—as the number of force fired transitions will never improve.
- Third, note that some Petri nets might contain invisible transitions which, after firing, enable themselves once more. To prevent situations where firing sequences would grow to an infinite length, a threshold on the number of subsequent invisible firings can be added. Finally, with this threshold, it is possible to enable the force firing of invisible transitions as well, dropping the $\mathtt{enabled}(t, m)$ requirement in Algorithm 3.3. Without the subsequent invisible firing threshold, allowing the force firing of invisible transitions would certainly cause firing sequences to grow infinitely, as the presence of one invisible transition is enough to prevent the recursive construction function from stopping.

Heuristic Replay: One Step Look-forward This replay strategy constructs one single firing sequence which can be mapped to the trace σ. For each encountered positive event $\sigma_i \in \sigma$, several sets of candidate transitions are established, see Table 3.4, top to bottom row. In case a candidate set is found, a transition is chosen randomly from the set and fired (if the set only contains one transition, naturally that transition is chosen).

The benefit of this replay strategy is that no exhaustive state space exploration needs to be performed, greatly decreasing the run time required when replaying

Algorithm 3.3 Replay strategy: complete exploration. For clarity, the initial marking M_0 and Petri net (P, T, F) are left implicit in the notation of f, similarly, the firing sequence f, initial marking M_0 and Petri net (P, T, F) are left implicit in the notation of M_i.

```
Input: $(P, T, F)$, $M_0$ % Given Petri net and initial marking
Input: $\sigma$ % An event log trace
Input: $\mu$ % Transition-activity mapping $\mu : T \mapsto A_L \cup \{a_i, a_b\}$
Input: enabled : $(T \times M) \to \{True, False\}$ % Function denoting if a transition is enabled under a given marking
    (M is set of all possible markings)
Input: nextmarking : $(T \times M) \to M$ % Function returning the marking after (force) firing a given transition in
    a given marking
Output: Constructed set of firing sequences $F$
 1: $F := \emptyset$
 2: $f := \langle\rangle$
 3: $Recurse(F, f, M_0, \sigma_1)$ % Call recursive function
 4: return $F$
 5: function RECURSE($F$, $f$, $m$, $\sigma_i$)
 6:     % Determine candidate transitions
 7:     $CT := \{t \in T | \mu(t) = sigma_i \vee (a_i = \mu(t) \wedge enabled(t, m))\}$
 8:     for all $t \in CT$ do
 9:         $f := \langle f, t \rangle$ % Expand sequence
10:         if $i = |\sigma|$ then
11:             % Add to set of possible sequences
12:             $F := F \cup f$
13:         else if $\mu(t) = a_i$ then
14:             $Recurse(F, f, nextmarking(t, m), \sigma_i)$
15:         else
16:             $Recurse(F, f, nextmarking(t, m), \sigma_{i+1})$
17:         end if
18:     end for
19: end function
```

Table 3.4: Replay strategy: heuristic replay with one step look-forward. This replayer iterates over various candidate transition sets (top to bottom row).

MAPPED TO ACTIVITY? $\mu(t) = \sigma_i$	ENABLED? $enabled(t, m)$	FIRING ENABLES NEXT (CURRENT) ACTIVITY? $\exists t_{next} \in T : [\mu(t_n) = z \wedge enabled(t_n, nextmarking(t, m))]$	INVISIBLE? $\mu(t) = a_i$	OUTCOME
yes	yes	yes ($z = \sigma_{i+1}$)	no	Preferred normal fire
yes	yes	no ($z = \sigma_{i+1}$)	no	Normal fire
–	yes	yes ($z = \sigma_i$)	yes	Enabling invisible fire
–	yes	no ($z = \sigma_i$)	yes	Greedy invisible fire (optional)
yes	no	yes ($z = \sigma_{i+1}$)	no	Preferred forced fire
yes	no	no ($z = \sigma_{i+1}$)	no	Forced fire
no	yes	yes ($z = \sigma_i$)	no	Recovering model-only fire (optional)
–	no	yes ($z = \sigma_i$)	yes	Recovering model-only forced invisible fire (optional)
no	no	yes ($z = \sigma_i$)	no	Recovering model-only forced fire (optional)
no	no	no	no	Log-only move (no transition fired)

traces. Additionally, this replay strategy corresponds with how human end-users interpret process models. Note that Table 3.4 contains some optional candidate sets. First of all, it is possible to enable greedy firing of invisible tasks. In some cases, this can be beneficial for when a one step look-forward is not sufficient to find an invisible firing subsequence to enable the positive activity at hand. On the other hand, greedy firing of invisible can cause the constructed firing sequence to contain redundant invisible transitions, which are fired but do not actually help in enabling the next non-invisible transition at hand, which is still ultimately force fired. As such, in most cases, it is best to skip this candidate transition set (if long invisible subsequences exist which lead to the enabling of the next positive activity, it is better to use the next replay strategy, which performs an invisible path exploration). Second, note the presence of model-only moves. Considering these candidate transition sets brings the heuristic procedure closer to alignment based evaluation schemes [112], whilst still retaining scalability. Algorithm 3.4 formalizes this replay strategy.

Heuristic Replay: Invisible Path Exploration This replay strategy also constructs one single firing sequence which can be mapped to the trace σ. Algorithm 3.5 provides a concise formalized overview of the workings of this replay strategy. This replay strategy also implements a heuristic approach, but contrary to the earlier one step look-forward technique, this approach attempts to explore "invisible paths" to immediately find the shortest invisible transition path necessary in order to fire the positive event at hand. Some optimizations can be implemented in order to improve the performance of function `FindShortest` in Algorithm 3.5. First, a limit can be set on the "depth" of the search, discarding paths with more than a certain number of steps, as it can be argued that the quality of process models requiring many invisible transition executions is lower than counterparts who do not require many silent executions. Next, it is possible to implement the search procedure in a depth first or breadth first manner. The optimal strategy is to combine the two and to perform a breadth first search first for a small amount of steps (say, paths of size three) before performing a deep exploration with depth first search. This approach is comparable to the replay strategy discussed in [99, 100]. Note that, for the multitude of Petri nets, invisible transition execution paths preceding the execution of a non-invisible transition will normally not be of great length, as even for Petri nets containing a multitude of routing invisible transitions, one invisible transition execution often suffices in order to reach the desired non-invisible transition, see for example the *connectedModel* in Figure 3.1.

Algorithm 3.4 Replay strategy: heuristic replay with one step look-forward.

```
Input: $(P, T, F)$, $M_0$ % Given Petri net and initial marking
Input: $\sigma$ % An event log trace
Input: $\mu$ % Transition-activity mapping $\mu : T \mapsto A_L \cup \{a_i, a_b\}$
Input: enabled : $(T \times M) \rightarrow \{True, False\}$ % Function denoting if a transition is enabled under a given marking
    ($M$ is set of all possible markings)
Input: nextmarking : $(T \times M) \rightarrow M$ % Function returning the marking after (force) firing a given transition in
    a given marking
Input: random : $T' \subseteq T \rightarrow T$ % Function returning random transition from a given set of transitions
Output: Constructed firing sequences $f$
 1: $f := \langle\rangle$
 2: $i := 1$
 3: $m := M_0$
 4: while $i \leqslant |\sigma|$ do
 5:     % Construct candidate transition sets
 6:     $MT := \{t \in T | \mu(t) = \sigma_i\}$, $ET := \{t \in T | enabled(t, m)\}$
 7:     $VT := \{t \in T | \mu(t) \neq a_i\}$, $IT := \{t \in T | \mu(t) = a_i\}$
 8:     $CET := \{t \in T | \exists t_n \in T : [\mu(t_n) = \sigma_i \wedge enabled(t_n, nextmarking(t, m))]\}$
 9:     $NET := \{t \in T | \exists t_n \in T : [\mu(t_n) = \sigma_{i+1} \wedge enabled(t_n, nextmarking(t, m))]\}$
10:     % Determine transition
11:     if $|MT \cap ET \cap NET| > 0$ then
12:         $t := random(MT \cap ET \cap NET)$
13:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
14:         $i := i + 1$
15:     else if $|MT \cap ET| > 0$ then
16:         $t := random(MT \cap ET)$
17:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
18:         $i := i + 1$
19:     else if $|IT \cap ET \cap CET| > 0$ then
20:         $t := random(IT \cap ET \cap CET)$
21:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
22:     else if $|MT \cap NET| > 0$ then
23:         $t := random(MT \cap NET)$
24:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
25:         $i := i + 1$
26:     else if $|MT| > 0$ then
27:         $t := random(MT)$
28:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
29:         $i := i + 1$
30:     else if $|VT \cap ET \cap CET| > 0$ then
31:         $t := random(VT \cap ET \cap CET)$
32:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
33:         $i := i + 1$
34:     else if $|IT \cap CET| > 0$ then
35:         $t := random(IT \cap CET)$
36:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
37:     else if $|VT \cap CET| > 0$ then
38:         $t := random(VT \cap CET)$
39:         $f := \langle f, t\rangle$, $m := nextmarking(t, m)$
40:         $i := i + 1$
41:     else
42:         % No transition fired
43:         $i := i + 1$
44:     end if
45: end while
46: return $f$
```

Algorithm 3.5 Replay strategy: heuristic replay with invisible path exploration.

```
Input: $(P,T,F)$, $M_0$ % Given Petri net and initial marking
Input: $\sigma$ % An event log trace
Input: $\mu$ % Transition-activity mapping $\mu : T \mapsto A_L \cup \{a_i, a_b\}$
Input: enabled : $(T \times M) \to \{True, False\}$ % Function denoting if a transition is enabled under a given marking
    (M is set of all possible markings)
Input: nextmarking : $(T \times M) \to M$ % Function returning the marking after (force) firing a given transition in
    a given marking
Input: random : $T' \subseteq T \to T$ % Function returning random transition from a given set of transitions
Output: Constructed firing sequences $f$
 1: $f := \langle\rangle$
 2: $m := M_0$
 3: for all $\sigma_i \in \sigma$ do
 4:     % Find the shortest sequence of invisible transitions in order to enable a transition mapped to $\sigma_i$. The
        transition mapped to $\sigma_i$ is concatenated at the end. If a transition mapped to $\sigma_i$ is already enabled under the
        current marking, the shortest sequence just contains this single transition without any invisible transitions.
        When no path can be found to enable a transition mapped to $\sigma_i$, the sequence is empty.
 5:     $in := FindShortest(m, \sigma_i)$
 6:     if $in = \emptyset$ then
 7:         $t := random(t \in T | \mu(t) = \sigma_i)$
 8:         $f := \langle f, t\rangle$
 9:         $m := nextmarking(t, m)$
10:     else
11:         $f := \langle f, in\rangle$
12:         $\forall t \in in : [m := nextmarking(t, m)]$
13:     end if
14: end for
15: return $f$
```

3.6.1.3 Evaluation Schemes

After having outlined different strategies to construct a firing sequence over a trace containing positive events, we now turn our attention to the evaluation of negative events. As discussed earlier, we make a difference between *state-bound* evaluation schemes, which evaluate negative events given the state of the model obtained after firing the last-seen positive events and *state-free* schemes, which evaluates negative events independently from the decision chain of transitions made before the occurrence of the negative event.

State-bound: First Marking Only Let us consider the following trace and associated firing sequence, found by applying one of the aforementioned replay strategies:

$f^{M_0,(P,T,F)}$	a	inv_1	inv_2	inv_3	b	c	d
σ	a	$-$	$-$	$-$	b	c	d

.

We now wish to check whether the process model at hand permits the negative event n^-, inserted right before the execution of b in σ. A first way to so do is to take the marking obtained after firing a (i.e. M_1) and check whether a transition mapped to n is enabled in this marking. If so, then the model should be punished on its precision dimension. This is the approach followed in [2]. Variations on this evaluation scheme include: "Last Marking Only", which in this case would check if a transition mapped to n is enabled given the marking reached after firing inv_3, i.e. right before firing b and "First and Last Marking Only", which checks both markings. However, as was also discussed before, we have to consider invisible transition paths as well in order to determine whether to process model allows to fire n after having fired a. A first possible approach towards doing so is to assume the invisible transition path as followed by the constructed firing sequence as fixed and bind the evaluation of negative events to this path.

State-bound: Using Constructed Invisible Transition Path We consider again the same trace and firing sequence. Instead of checking whether negative event n^- is enabled by binding the evaluation to a single marking, we now check each marking visited by the constructed firing sequence starting from the marking obtained after firing the last positive event at hand and before firing the next positive event, that is, the markings M_1, M_2, M_3, M_4 obtained after firing a, inv_1, inv_2 and inv_3 respectively. If there exists a transition mapped to n which is enabled under one of these visited markings, the model is punished on its precision dimension. Note that it is possible to construct a "lenient" form of this check where an enabled transition mapped to n should exist for *each* marking visited. As this performs a very weak check where only very imprecise models will be punished, this approach is not recommended.

State-bound: Using Free Invisible Transition Path The last state-bound negative event evaluation technique also assumes a constructed firing sequence, but now starts from the marking obtained after firing the last-seen positive event (M_1 after a) and explores all invisible transition paths to see whether the model is able to enable to firing of n^-, using the same $\mathtt{FindShortest}$ function as was also applied in Algorithm 3.5. This evaluation scheme thus *no longer binds itself to the invisible path used to reach the next positive event*, as we effectively desire to find an answer to the question: "After firing the last positive event, would the model have been able to execute the following negative event?" Since the next positive event is not yet known at this time, the model is free to perform an invisible path exploration to answer this query. However, since such an exploration requires more computational time, users might nevertheless choose to bind the evaluation to the firing sequence and its invisible firings.

State-free Evaluation Situations might exist where the decision to fire an invisible transition leading to the enabling of a negative event under consideration (n^- before b) should be made before encountering the last seen positive event (i.e. before a). For example, an invisible transition inv_0 might have been enabled in the starting marking, which was not included in the firing sequence (as a was also enabled in the starting marking), but might have enabled transitions for both a and n^- after firing a). None of the state-bound approaches are able to uncover these cases. To perform a state-free evaluation, *a new trace* is constructed consisting of the positive event prefix up until the negative event with the negative event concatenated at the end, i.e. $\langle a, n \rangle$ in this example. This trace is then replayed on the process model (using one of the replay strategies outlined above) to see whether it is able to replay this "negative trace". Note however that end-users do not often think to evaluate negative events in this manner, as the decision to fire the previous positive event is already made by the time the negative event arrives, and the future of running traces is not known at execution-time.

3.6.1.4 Dealing with Forced Firings

It should be noted that it is hard to check negative events using the state-free evaluation approach when the positive event prefix so far has caused transitions to be force fired, or when allowing transitions to be force fired to determine whether n ever becomes enabled. That is, if force firing would be allowed in order to determine whether a negative event (following a positive prefix) is accepted by the process model or not, it would follow that many negative events will be accepted (thus lowering the ultimate precision score), as the model will attempt to force fire *any* possible transition which leads to the enabling of the (final) negative event. Therefore, we do not permit force firing when evaluating negative events in this manner.

However, by not doing so, it can be argued that the heuristic state-bound approaches described above suffer from a similar limitation: *as the force firing of transitions also produces additional tokens which may subsequently lead to the enabling of negative events.* Other evaluation approaches found in literature also encounter this issue. In [2], for example, the choice is made to allow this. Force firing is thereby accepted as an intrinsic part of Petri net semantics and should thus be taken into account when evaluating precision and generalization; the authors argue that this compensates for the remaining tokens which are not punished by the fitness metric after replay. In [105], on the other hand, traces are only evaluated in terms of precision up until the point where the first force firing occurs.

Note that not all force fired transitions lead indefinitely to an "unstable" state until the end of the trace is reached, although determining a method to resolve these instabilities for more complex cases is a non-trivial problem. Therefore, we list the following options to deal with force fired transitions for state-bound evaluation approaches (the state-free evaluation disallows force firing when evaluating negative events): stop the evaluation of negative events after a force fire; only check negative events which do not directly follow a positive event for which a transition was force fired; or check all negative events nevertheless the force firings executed.

Adriansyah et al. [108–112] have proposed a replay technique which avoids force firing altogether by allowing the process model and log to move independently, resulting in a so-called model-log alignment. This technique can be regarded as a fourth possible replay strategy. Using this approach, we evaluate positive events based on whether the event under consideration could be successfully aligned with a transition execution in the process model (opposite to checking positive events based on whether they were force fired or not as with the other replay strategies). Negative events can be evaluated using the evaluation techniques as described above, starting from the marking obtained after the last aligned model-log move.

3.6.1.5 Experimental Results

In order to assess the impact of the discussed replay strategies, negative event evaluation schemes and manners to deal with force firings, we set up an experiment using the process models and event log as shown in Figure 3.1. We evaluate the fitness, precision and generalization capabilities of these models using our proposed *Behavioral Recall* ($\mathrm{r_B}$), *Behavioral Precision* ($\mathrm{p_B}$) and *Behavioral Generalization* ($\mathrm{g_B}$) metrics. Note that we *do not weight the generated negative events* as to clearly assess the impact of the various replay strategies and negative event evaluation schemes on conformance checking results. We evaluate these models (for which the desired quality assessment is known beforehand) over an exhaustive set of parameter combinations. Expected fitness ("F+/-"), precision ("P+/-") and generalization ("G+/-") are depicted for each model:

- For the *perfectModel*, the results in Table 3.5 show that each configuration vector returns the same result in this case, with maximal recall, precision and generalization scores, as desired.
- Since *singleModel* is not fitting (see results in Table 3.6), the impact of the different ways to handle force fired transitions becomes apparent here. It is also interesting to note that the alignment based replay strategy avoids force firing by allowing the model and the log to move independently, but this increased amount of "non-aligned" moves causes that the model is punished more on the dimensions of recall and precision. We cannot observe a difference between different replay strategies or negative event evaluation schemes for this model.
- The results in Table 3.7 show that the *flowerModel* can be easily and correctly be evaluated in terms of fitness, precision and generalization. All configurations return the same values.
- The *stackedModel* illustrates an important difference between state-bound and state-free evaluation techniques (see Table 3.8), which is especially visible on generalization values as shown above. State-bound evaluation techniques consider the currently-reached marking as fixed and will not backtrack through the execution history to investigate whether different choices might have led to different results. State-free techniques on the other hand do start from the initial marking, with the goal of investigating whether a negative event could be accepted by the process model or not, following a certain activity sequence. Since the *stackedModel* binds the user to a certain path from the beginning, the ability to generalize will thus be negatively impacted by this binding. In most cases, the state-bound perspective will be preferred in evaluation experiments such as these, although the ability thus exist to opt for a state-free approach as well, where each negative event will be queried independently from the current context. In addition, we also note a clear difference in the performance of the various replay strategies in this case, as heuristic approaches are not able to determine the correct path to follow to satisfy the remainder of the trace at the beginning of trace replay. Still, this is not so much a weak point of heuristic approaches, as it can be said that in an execution environment

(i.e. a running system), it is also infeasible to determine which decision to choice without knowing the future behavior of the trace at hand.

- The *connectedModel* is comparable with a flower model in terms of quality (see Table 3.9). Just as with the *flowerModel*, a configuration vector is easily found which correctly returns the expected values. However, remark here the difference between the various negative event evaluation schemes. Since the lenient evaluation strategies consider the transition sequence between the positive events as given, this approach will fail to detect any precision errors, since all of them require an investigation of "invisible" paths outside the one actually followed to execute the next positive event. Finally, another interesting remark is the fact that event heuristic replay strategies remain able to correctly assess *connectedModel*'s on the dimension of fitness, even although so many invisible transitions are present. This shows that, ever for complex Petri nets, a one look-forward heuristic approach is oftentimes sufficient to correctly deduce the correct invisible transition(s) to fire.

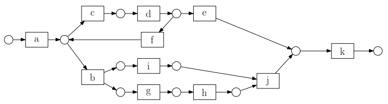
3.6.1.6 Concluding Remarks

We have discussed in depth the problem of replaying event traces on Petri nets for sequences containing both positive and negative events, together with a description of different schemes to evaluate negative events in order to investigate the impact of these strategies and techniques on conformance checking metrics which make use of both positive and negative events to assess process models on the quality dimensions of recall, precision and generalization. After having performed a thorough experimental evaluation of the discussed replay strategies and negative event evaluations schemes, it was decided to include two different approaches in the conformance checking framework described in this chapter:

1. First, a heuristic replayer performing a one look-forward investigation, as was detailed in this section.
2. An alignment based strategy as proposed by [108–110].

In both cases, negative events are evaluated by performing a state-bound full invisible path exploration (i.e. not bound to the invisible path chosen for the positive event firing sequence), ignoring negative events directly following a positive

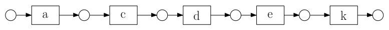
Table 3.5: Impact of various replay strategies and negative event evaluation schemes for *perfectModel*.



Model: *perfectModel* (Reference Model):
F+, P+, G+

Replayer	Evaluation	Forced Fire Handling	r_B	p_B	g_B
One-step Lookforward (Conservative)	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-free	n/a		1.00	1.00
One-step Lookforward	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-free	n/a		1.00	1.00
Invisible Path Exploration	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-free	n/a		1.00	1.00
Complete Exploration	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-free	n/a		1.00	1.00
Alignment Based	n/a	n/a	1.00	1.00	1.00

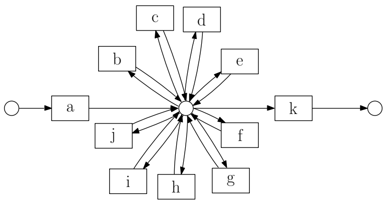
Table 3.6: Impact of various replay strategies and negative event evaluation schemes for *singleModel*.



Model: *singleModel* (Single Path Model):
F-, P+, G-

Replayer	Evaluation	Forced Fire Handling	r_B	p_B	g_B
One-step Lookforward (Conservative)	State-bound: First Marking	Check All	0.38	0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Constructed Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Free Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-free	n/a		1.00	0.00
One-step Lookforward	State-bound: First Marking	Check All	0.38	0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Constructed Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Free Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-free	n/a		1.00	0.00
Invisible Path Exploration	State-bound: First Marking	Check All	0.38	0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Constructed Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Free Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-free	n/a		1.00	0.00
Complete Exploration	State-bound: First Marking	Check All	0.38	0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Constructed Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-bound: Free Invisible Path	Check All		0.83	0.00
		Check Only After Non-forced		0.91	0.00
		Stop After Forced		1.00	0.00
	State-free	n/a		1.00	0.00
Alignment Based	n/a	n/a	0.45	0.46	0.04

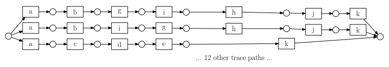
Table 3.7: Impact of various replay strategies and negative event evaluation schemes for *flowerModel*.



Model: *flowerModel* (Flower Model):
F+, P-, G+

Replayer	Evaluation	Forced Fire Handling	r_B	p_B	g_B
One-step Lookforward (Conservative)	State-bound: First Marking	Check All	1.00	0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Constructed Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Free Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-free	n/a		0.12	1.00
One-step Lookforward	State-bound: First Marking	Check All	1.00	0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Constructed Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Free Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-free	n/a		0.12	1.00
Invisible Path Exploration	State-bound: First Marking	Check All	1.00	0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Constructed Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Free Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-free	n/a		0.12	1.00
Complete Exploration	State-bound: First Marking	Check All	1.00	0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Constructed Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-bound: Free Invisible Path	Check All		0.12	1.00
		Check Only After Non-forced		0.12	1.00
		Stop After Forced		0.12	1.00
	State-free	n/a		0.12	1.00
Alignment Based	n/a	n/a	1.00	0.12	1.00

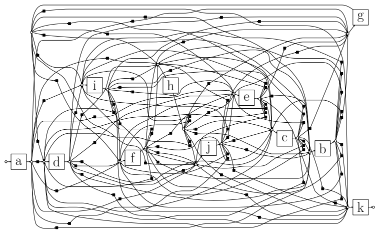
Table 3.8: Impact of various replay strategies and negative event evaluation schemes for *stackedModel*.



Model: *stackedModel* (Stacked Model):
F+, P+, G-

Replayer	Evaluation	Forced Fire Handling	r_B	p_B	g_B
One-step Lookforward (Conservative)	State-bound: First Marking	Check All	0.84	0.54	0.05
		Check Only After Non-forced		0.79	0.01
		Stop After Forced		1.00	0.01
	State-bound: Constructed Invisible Path	Check All		0.52	0.13
		Check Only After Non-forced		0.86	0.13
		Stop After Forced		1.00	0.00
	State-bound: Free Invisible Path	Check All		0.52	0.17
		Check Only After Non-forced		0.69	0.02
		Stop After Forced		1.00	0.01
	State-free	n/a		1.00	1.00
One-step Lookforward	State-bound: First Marking	Check All	0.84	0.55	0.10
		Check Only After Non-forced		0.70	0.00
		Stop After Forced		1.00	0.00
	State-bound: Constructed Invisible Path	Check All		0.55	0.11
		Check Only After Non-forced		0.67	0.00
		Stop After Forced		1.00	0.02
	State-bound: Free Invisible Path	Check All		0.50	0.11
		Check Only After Non-forced		0.81	0.00
		Stop After Forced		1.00	0.01
	State-free	n/a		1.00	1.00
Invisible Path Exploration	State-bound: First Marking	Check All	0.84	0.71	0.22
		Check Only After Non-forced		0.81	0.37
		Stop After Forced		1.00	0.38
	State-bound: Constructed Invisible Path	Check All		0.62	0.40
		Check Only After Non-forced		0.83	0.46
		Stop After Forced		1.00	0.32
	State-bound: Free Invisible Path	Check All		0.69	0.34
		Check Only After Non-forced		0.77	0.33
		Stop After Forced		1.00	0.26
	State-free	n/a		1.00	1.00
Complete Exploration	State-bound: First Marking	Check All	1.00	1.00	0.00
		Check Only After Non-forced		1.00	0.00
		Stop After Forced		1.00	0.00
	State-bound: Constructed Invisible Path	Check All		1.00	0.00
		Check Only After Non-forced		1.00	0.00
		Stop After Forced		1.00	0.00
	State-bound: Free Invisible Path	Check All		1.00	0.00
		Check Only After Non-forced		1.00	0.00
		Stop After Forced		1.00	0.00
	State-free	n/a		1.00	1.00
Alignment Based	n/a	n/a	1.00	1.00	0.00

Table 3.9: Impact of various replay strategies and negative event evaluation schemes for *connectedModel*.



Model: *connectedModel* (Fully Connected Model):
F+, P-, G+

Replayer	Evaluation	Forced Fire Handling	r_B	p_B	g_B
One-step Lookforward (Conservative)	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		0.13	1.00
		Check Only After Non-forced		0.13	1.00
		Stop After Forced		0.13	1.00
	State-free	n/a		0.13	1.00
One-step Lookforward	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		0.13	1.00
		Check Only After Non-forced		0.13	1.00
		Stop After Forced		0.13	1.00
	State-free	n/a		0.13	1.00
Invisible Path Exploration	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		0.13	1.00
		Check Only After Non-forced		0.13	1.00
		Stop After Forced		0.13	1.00
	State-free	n/a		0.13	1.00
Complete Exploration	State-bound: First Marking	Check All	1.00	1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Constructed Invisible Path	Check All		1.00	1.00
		Check Only After Non-forced		1.00	1.00
		Stop After Forced		1.00	1.00
	State-bound: Free Invisible Path	Check All		0.13	1.00
		Check Only After Non-forced		0.13	1.00
		Stop After Forced		0.13	1.00
	State-free	n/a		0.13	1.00
Alignment Based	n/a	n/a	1.00	0.13	1.00

event for which a transition was force fired. Our experiments show that these approaches offer the optimal tradeoff between understandability, speed and accuracy.

3.6.2 Additional Configuration Parameters

This section lists some additional configuration parameters which were included in our implementation. They are meant to be used by expert end-users and added for completeness. In the experimental evaluation (Section 3.5), none of these additional configuration parameters were considered:

3.6.2.1 Alternative Window Ratio

An alternative window ratio for assessing the impact of a generalized event. By default, the impact of a negative event on generalization is equal to:
$1 - \mathtt{Min}_{\forall \textit{window comparisons}}(\frac{|\textit{window}| - |\textit{matching window}|}{|\textit{window}|})$ or, also:
$\mathtt{Max}_{\forall \textit{window comparisons}}(\frac{|\textit{matching window}|}{|\textit{window}|})$ i.e. the *maximal matching window ratio* found over all window comparisons performed. We can also define an alternative ratio, the *minimal matching window ration* which is defined as:
$\mathtt{Min}_{\forall \textit{window comparisons}}(\frac{|\textit{matching window}|}{|\textit{window}|})$, so that the contribution of a negative event to generalization is no longer the direct inverse of its contribution to precision. In general, enabling this option causes the weight of a generalized event to drop, meaning that a less strict evaluation will be performed in terms of generalization. Note that this is a valid strategy towards making the *Weighted Behavioral Generalization* metric more lenient.

3.6.2.2 Cutted Windows

By default, the complete window before the candidate negative event is considered for comparison. However, it is also possible to only consider the window before an activity equal to the positive activity at the current position is encountered. For example, when we aim to check the candidate negative event n^- in a $\sigma = \langle a, b, c, a, b, c, a, b, \{n^-\}, c\rangle$, the window stemming from this would be equal to $\langle a, b\rangle$ and not $\langle a, b, c, a, b, c, a, b\rangle$. This makes the window comparison more lenient.

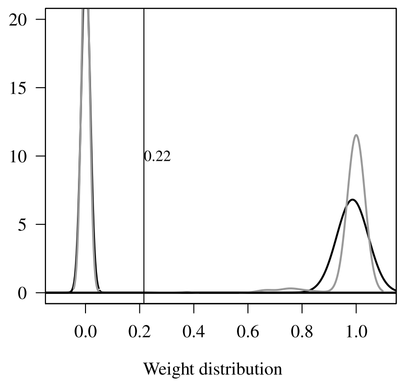
In general, enabling this option causes the weight of a negative event to drop, meaning that a less strict evaluation will be performed in terms of precision. In most event logs, the difference is negligible, however, but users might wish to consider this option when the event log at hand contains recurrent behavior and very long traces.

3.6.2.3 Configurable Window Sizes

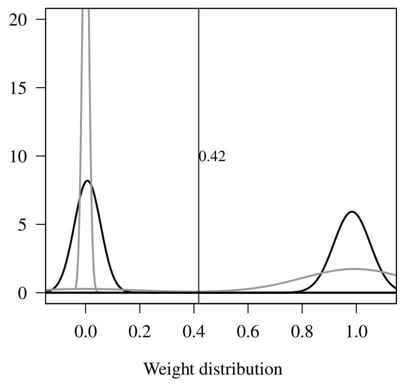
By default, the complete window before the candidate negative event is considered for comparison. However, like in the original algorithm, it is also possible to bind the size of the window. This can be configured both for the calculation of a negative event's impact on precision and generalization.
In general, lowering the size of the window causes the weight of a negative event to drop, meaning that a less strict evaluation will be performed in terms of precision. In most cases, we recommend to keep the window sizes as is, i.e. unlimited size, as the weighting ratios itself are sufficient to deal with the problem of event log completeness, as was illustrated before.

3.6.2.4 Weight Threshold

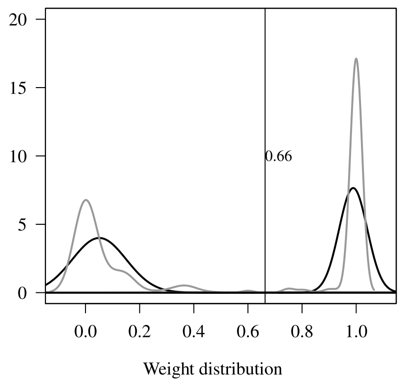
Whereas currently all negative events are taken into account for the calculation of the precision and generalization metrics, it is also possible to allow the user to supply a threshold based on which negative events can be discarded when evaluating precision and generalization (e.g. consider only negative events with a weight above 0.8 to calculate precision). As an interesting remark, we note that it would be possible to offer a threshold suggestion by fitting a two gaussian mixture model on the distribution of weights as generated for all negative events. Figure 3.14 shows how such an approach would work in practice.
In most cases, this is not necessary, however, as the weights themselves are robust enough so that no additional filtering is needed.



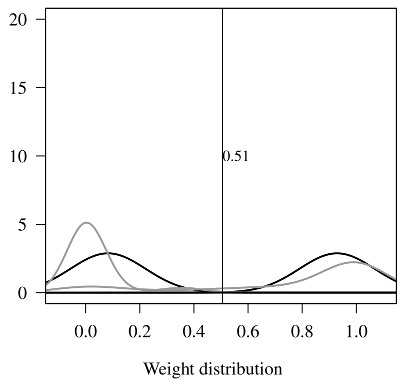
Event log *a5*.



Event log *a5* (with noise).



Event log *driversLicenseLoop*.



Event log *driversLicenseLoop* (with noise).

Figure 3.14: Derived cutoff threshold for distinguishing between (in)correct negative events as derived by a two gaussian mixture model. The gray lines indicate density plots of the negative events based on their true status, whereas the black lines denote the two fitted gaussians, with the overlap point indicated as a vertical line. Note that the "true" status is shown only for completeness; naturally, it is not taken into consideration when deriving the threshold.

3.6.3 Alternative Generation Strategies

The generation of negative events for a position in a trace considers the prefix of the trace up to this position. Readers might wonder why it is sufficient to consider only the prefix up to a position and to neglect the suffix. In addition, readers might also wonder if it is possible to consider less strict prefixes (and suffixes) to drive the generation of negative events. In principle, all of the following options are possible, in line with the transition system discovery algorithm presented in [79]:

- Ordered sequence, i.e. a window. This is what is done in our induction approach.
- A bag: a multiset of activities having occurred before the candidate negative event.
- A set of activities having occurred before the candidate negative event.

First, to tackle the issue of considering suffixes, in running process support information systems, it is rather uncommon that one is able to look into the future to see which event executions are coming up. As such, the ability of a process model to parse a positive or negative event (i.e. any attempt at executing a task) can be based on historical information only. Put in more technical terms: an execution history is sufficient (together with an initial marking) to bring a process model in a state or set of states from which the set of enabled and non-enabled tasks can be derived. Concerning *post-hoc replay*, however, it is true that considering the full trace can help to determine the most optimal path throughout the process model (this is done for example by the alignment based replay techniques), but we argue that actual process model execution semantics should not be determined based on future events. Note also that when an event log would be absolutely complete, the prefixes are also sufficient to generate all negative events, without considering the suffix.

Considering other prefix representations, we have performed an experiment where negative events are induced based on a bag representation (allowing freedom in terms of ordering), with the weight of a negative event based on the "minimum unmatching bag ratio", i.e. one minus the fraction of activities in the current positive's event window also present in the comparison window at hand). Table 3.10 lists the results of this alternative approach for the artificial event logs included in the experimental evaluation of this chapter.

From the results, we can conclude that the activity bag based approach is unable to return a good precision/generalization measure in many cases. Although it is possible to experiment with other mixed representation schemes, i.e. an ordered set, we conclude that the approach based on event windows performs robustly and correctly in practice and is thus put forward as the recommended generation strategy.

Table 3.10: Comparison of window prefix based weighted artificial negative event generation with activity bag based induction strategy.

Event Log	p_B^w (heuristic)	p_B^w (alignment)	g_B^w (heuristic)	g_B^w (alignment)	p_B^w (bags; heuristic)	p_B^w (bags; alignment)	g_B^w (bags; heuristic)	p_B^w (bags; alignment)
a10skip	1.00	1.00	0.91	0.91	1.00	1.00	0.05	0.05
a12	1.00	1.00	0.83	0.83	1.00	1.00	0.04	0.04
a5	1.00	1.00	0.88	0.88	1.00	1.00	0.16	0.16
a6nfc	0.94	1.00	0.61	0.61	0.98	1.00	0.05	0.04
a7	1.00	1.00	0.70	0.70	1.00	1.00	0.16	0.16
a8	1.00	1.00	0.94	0.94	1.00	1.00	0.08	0.08
betaSimplified	1.00	1.00	0.70	0.70	1.00	1.00	0.02	0.02
choice	1.00	1.00	1.00	1.00	1.00	1.00	0.07	0.07
complex	0.73	0.74	0.51	0.51	1.00	1.00	0.09	0.09
driversLicense	1.00	1.00	0.80	0.80	1.00	1.00	0.03	0.03
driversLicenseLoop	0.97	0.97	0.83	0.83	1.00	1.00	0.06	0.06
herbstFig3p4	0.99	0.99	0.94	0.94	1.00	1.00	0.05	0.05
herbstFig5p19	1.00	1.00	0.77	0.77	1.00	1.00	0.10	0.10
herbstFig6p18	0.97	0.97	1.00	1.00	1.00	1.00	0.12	0.12
herbstFig6p31	1.00	1.00	0.57	0.57	1.00	1.00	0.10	0.10
herbstFig6p36	1.00	1.00	0.57	0.57	1.00	1.00	0.02	0.02
herbstFig6p38	1.00	1.00	0.40	0.50	1.00	1.00	0.08	0.06
herbstFig6p41	1.00	1.00	0.73	0.73	1.00	1.00	0.03	0.03
l2l	1.00	1.00	1.00	1.00	1.00	1.00	0.10	0.10
l2lOptional	1.00	1.00	1.00	1.00	1.00	1.00	0.12	0.12
l2lSkip	1.00	1.00	1.00	1.00	1.00	1.00	0.08	0.08
confDimPerfect	1.00	1.00	0.84	0.84	1.00	1.00	0.07	0.07
confDimConnected	0.13	0.13	1.00	1.00	0.38	0.38	0.88	0.88
confDimFlower	0.12	0.12	1.00	1.00	0.36	0.36	1.00	1.00
confDimSingle	0.41	0.46	0.23	0.03	0.57	1.00	0.06	0.09
confDimStacked	0.70	1.00	0.36	0.00	0.90	1.00	0.08	0.00

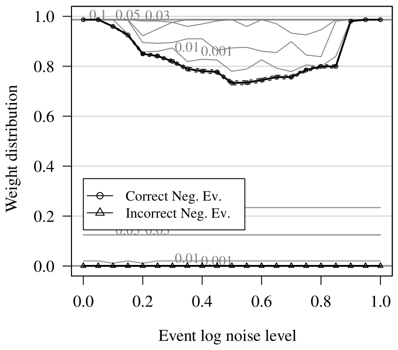
3.6.4 Dealing with Noise

Our proposed weighting scheme for negative events ignores how often a certain trace has been observed in the log. This means that, in the extreme case, a weight of 0 of a negative event means that there is a trace with the same prefix up until an activity equal to the candidate negative under consideration. However, it does not matter whether one or hundreds of such traces have been observed in the log. In cases where absence of noise is assumed, this poses no issue, as the frequency of traces is still taken into account when performing the actual evaluation on the event log. In cases where noise is present, however, it might be the case that this 0 weighting follows from a single, noisy trace.

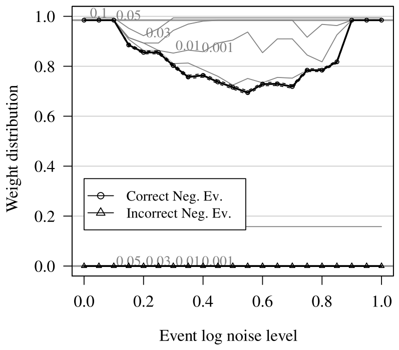
The aspect of noise in event logs is very broad and can involve many different types of deviations, ranging from missing events, swapped events, replaced events or inserted events. For many of these, frequency based, statistical approaches can be developed which can serve to "filter" the event log before considering it in further analysis steps, whether it be process discovery or induction of negative events. As such, we have chosen to avoid the aspect of noise in the negative induction algorithm and consider noise to be an issue which needs to be dealt with using a separate class of algorithms before any other analysis steps are performed, were it can be assumed that the event log is free of disturbing noise.

As an exercise, we might, however, consider the aspect of trace frequencies to determine the weight of a negative event. The main challenge here is that it is not quite trivial to propose a threshold or modified weighting mechanism which does take into account the trace frequency when deciding on the score given to a negative event, while also remaining robust to skewed or unbalanced trace frequency distributions (many event logs contain a small subset of frequently occurring trace variants, next to many low-frequent variants—the "long tail"). As such, we set up an experiment—using the same artificial event logs as were utilized in Section 3.5—were we once again calculate negative event weights, this time over varying sizes of log noisiness, denoted as a percentage of events which were tampered with (swapped, removed, inserted). Once again, it is crucial to emphasize that, to generate the artificial negative events, the input log in which negative events are induced is kept constant over all runs, corresponding with the logs listed in Table 3.1, whereas the log used to build the suffix tree is modified for each run and set equal to the different noisy logs *together* with the original, non-noisy log. Even more than the completeness robustness check performed in

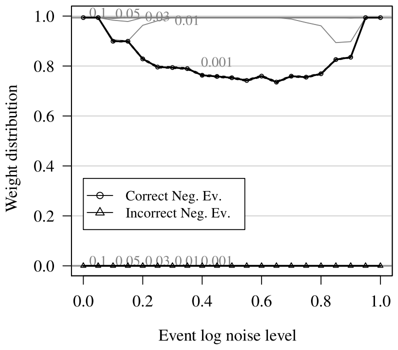
Subection 3.3.3, this is important, as we are only able to derive the “true status” of the negative events (based on the associated Petri net) when the event log is fitting, which cannot be guaranteed in the case of noisy logs. The reason why the non-noisy log is included is because we are interested in how the weightings of negative events change due to the effects of noisy data. If we would induce negative events in the original logs based on the noisy logs only, many incorrect negative events would receive a higher weighting as no matching windows of sufficient length can be found in the noisy event log. Normally, negative events are induced in a log based on an event log which is equally or more complete than the original log, i.e. the log itself. Here, we are interested in the *ceteris paribus* effects of noisy data, as the end goal of the experiment is to analyze how the weightings evolve over these noise levels.



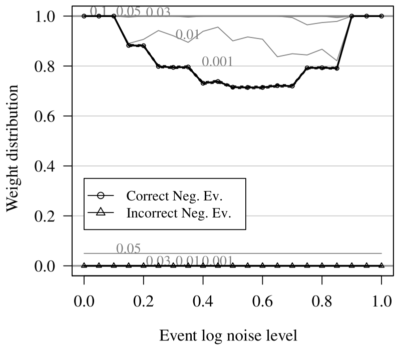
Event log *a5*.



Event log *a6nfc*.

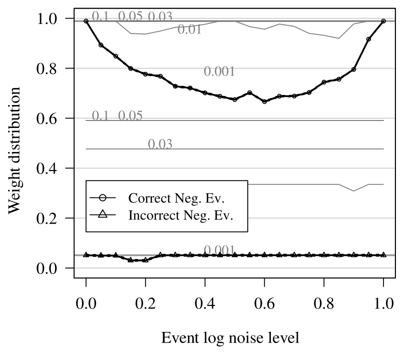


Event log *betaSimplified*.

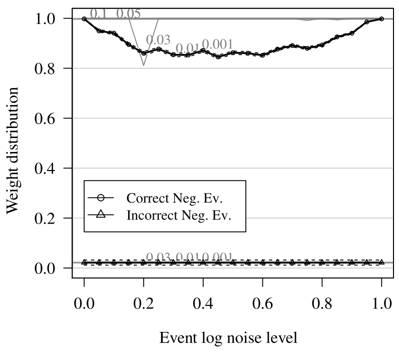


Event log *choice*.

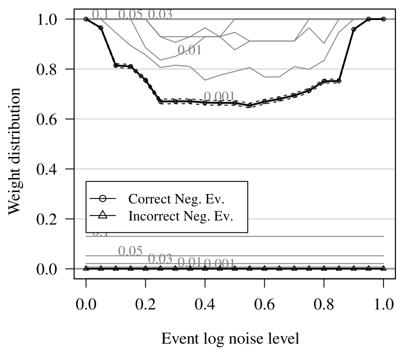
Figure 3.15: Evolution of artificial negative event weights in comparison with event log noisiness. The gray lines denote weights after infrequent traces were removed in the event log used to build the suffix tree.



Event log *driversLicenseLoop*.



Event log *herbstFig3p4*.



Event log *l2lOptional*.

Figure 3.15 (continued): Evolution of artificial negative event weights in comparison with event log completeness.

The results of the experiment are presented in Figure 3.15. The following observations can be made. First, note that the weights of generated negative events indeed drop as more noise is added, but the effects are relatively limited (negative events never drop to a weight of zero, as this would only occur in very unlikely circumstances), and reach a tipping point after a 50% noise level. This is to be expected, as after this level the event log becomes so noise that the—now almost random—traces become useless again towards disproving a candidate negative event, thus causing the weights to rise again.

Next, note the addition of the gray lines in Figure 3.15. These represent various threshold levels n, which is used to modify the event log used to build the suffix tree (the original and noisy log, as we recall) so that infrequent traces are removed. A trace is infrequent whenever its multiplicity m is $m \leqslant n \times |L|$ with

$|L|$ the size of the original log. This procedure is applied to the complete suffix tree building log (original and noisy), however, as we again are interested in seeing the effects of this filtering step *ceteris paribus*. The results suggest that 0.03 can be put forward as a good threshold to prevent the dropping of negative event weights under noise, but we also emphasize the fact that this threshold is highly dependent on the variance structure of the event log. In some cases—when choosing a threshold which is too high, especially—we also see that the weights for incorrect negative events rise from zero to a higher number, which is definitely an undesirable scenario. As such, we reiterate our previous statement that we regard the aspect of noise in event logs as a separate topic, for which specialized techniques should be applied. We have proven, however, that for small amounts of noise, the weightings of the negative events remain stable enough to be used as-is in subsequent steps (e.g. conformance checking). The highest impact is observed in cases where an event log contains 50% noise or more, in which case we recommend users to obtain a better suited (less noisy) event log, as no single technique (conformance checking or process discovery) will be able to derive useful results from such input set.

3.7 Conclusions

In this chapter, a novel conformance checking approach was put forward to determine how well a process model performs in terms of precision and generalization. The validity of the approach was illustrated by evaluating the proposed artificial negative event weighting method and by performing an experimental setup to benchmark our metrics in comparison with related techniques.

Our approach offers the following contributions. First, we have outlined an improved strategy for artificial negative event induction, extending it with a weighting method in order to indicate the confidence given to a candidate negative event, which helps to deal with incomplete logs, and implementing it in a manner which greatly improves the scalability of the technique. Second, these weighted artificial negative events form the basis of two novel metrics for precision and generalization. The introduction of a new generalization metric is an important contribution, due to the fact that the metric does not rely on a probabilistic estimator. Third, it was illustrated how these metrics can be applied on Petri net models and we have implemented both a heuristic and log-model alignment based approach in order to perform trace replay. All described techniques and metrics are made available in a number of ProM plugins, being the first implementation to provide a full conformance checking framework using negative events.

In closing, an opportunity for future work is put forward: although the *Weighted Behavioral Generalization* (g_B^w) metric provides a solid first step towards determining a process model's ability to generalize, it was found that the metric performs a rather strict evaluation. This is mainly due to the fact that, currently, the usefulness of a negative event towards generalization is direct inversely related to its use towards precision. Thus, the same weighted negative event can be used towards assessing both precision and generalization. It would be reasonable to constrain this behavior in future additions.

Chapter 4

Artificial Negative Events as Unobserved Events

"Three things cannot be long hidden:
the sun, the moon, and the truth."
– Buddha

4.1 Introduction

In the previous chapters, we have outlined the concept of artificial negative events, together with various strategies to generate them based on a provided event log containing positive behavior only. Based on this, a novel conformance checking approach was put forward to determine how well a process model performs in terms of precision and generalization.

This chapter looks at the concept of artificial negative events from a slightly different viewpoint. That is, readers might notice that the concept of an artificial negative event differs from a natural negative event in the sense that a natural negative event was a recorded attempt to execute a certain activity which was prohibited (i.e. attempted in real-life), whereas a collection of artificial negative events represent behavior which was unobserved—based on a configurable completeness assumption which has been shown to be robust—which is subsequently being "accepted" as being undesirable behavior for a process model to exhibit.

However, when looking at a set of artificial negative events as being representations of unobserved behavior, other analysis possibilities can be put forward, which will be elaborated briefly in this chapter. The basic idea can be summarized as follows: by applying our implemented artificial negative event generation strategies, artificially inducing events based on process history provides a fitting tool in order to detect which events were never observed at certain times in certain process instances. By examining these "under-represented" events and matching them with the set of current business rules and designed process models, investigators and analysts can spot non-occurred events which were, however, allowed and perhaps even desired. Such cases provide a good indication to investigate further in order to uncover the root causes behind this missing behavior, especially if the absence of a certain set or sequence of events is overshadowed by the presence of another set (meaning that, for some reason, certain perfectly valid alternatives are more or less ignored during day-to-day business conduct in favor of other activity paths). Applying the induction technique as described above thus allows us to uncover and analyze another group of events in our classification framework, namely those that did not occur in real-life (representing rare or infrequent tasks): the unobserved events.

4.2 Uncovering Implementation Gaps

We define unobserved events as business events for which the business process analyst cannot find evidence of their existence and that are not recorded in the provided event log. Two main subcategories can be distinguished: unobserved events for which the occurrence is expected in normal mode of operation and therefore might imply a dangerous deviation, and unobserved events for which an occurrence was not expected, i.e. the normal case. The latter are mainly used for learning business rules from the behavior present in an event log, as will be illustrated in the next section.

For unobserved events which were nevertheless allowed according to normal business conduct or even expected in normal business operations, it can be observed that they reflect anomalies in the process model, alerting the business. The business may consider to provide training, impose new rules, or adjust the designed model to reality because of its obsolescence. Note that artificial negative event generation is not the only way to retrieve these cases, as they can also be retrieved by means of LTL property verification [128–131], although the latter requires the presence of some user or domain knowledge.

Consider for instance the following illustrative example. Suppose that the decision is made to record the execution of a "Discuss with client" activity by asking employees to register this activity in the supporting information system. This activity is modeled in some prescriptive process model, where it follows after an "Ask approval" activity and can be executed with a "Sign credit request" activity. After the IT department has been instructed to implement the change, the concept of unobserved events can be utilized to verify whether this check has been implemented correctly. First, a process model is extracted from the recorded event log, by using one of the many available process discovery techniques. Next, we induce artificial negative events to uncover all behavior that did not occur in reality in the event log, given a certain process instance history—note that the configuration parameters then serve as a control for strictness; in most cases, a window of size one can be already helpful to uncover missing but perhaps expected behavior. Next, all negative events for which is it known that their actual execution would violate a business rule are removed, as this can be considered to be correct unobserved behavior[1]. The remaining set of negative events then represents allowed unobserved behavior.

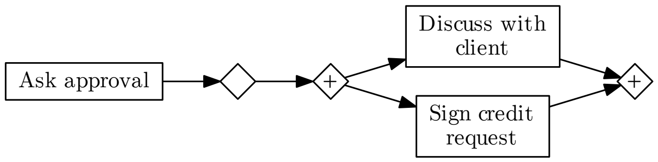
By annotating the discovered process map with this information, we can immediately inspect which activity flows are allowed in the process model but did not occur in the event log, see e.g. Figure 4.1. Analysts can then perform further investigations to uncover the root causes behind this fact. For our example, the parallel split between "Discuss with client" and "Sign credit request" was implemented incorrectly (modeled instead as a sequence in the information system), which is immediately highlighted when zooming in on the set of allowed unobserved events. Actions can be undertaken to investigate the causes of the absence of the events and measures may be applied to prevent this in the future.

4.3 Enhancing Declarative Process Models

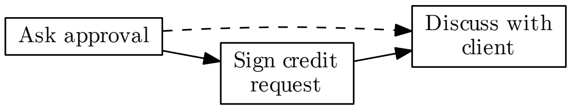
4.3.1 Introduction

Let us now focus our attention towards deriving a generalized approach to utilize unobserved events, namely in the context of the validation and enhancement of

[1]Note however that this set of unobserved behavior can also be helpful, i.e. to measure pre-set key performance indicators, as these unobserved events can be used to indicate that a preferred path is always followed, thus evidencing normal process execution.



Prescribed model, allowing parallelism between "*Discuss with client*" and "*Sign credit request*".



Discovered model only allows one execution order. Unobserved activity flow can be annotated to spot implementation issues.

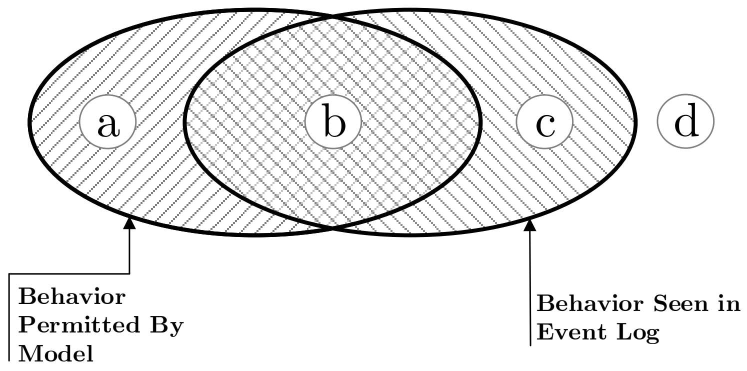
Figure 4.1: Unobserved events can help to highlight permitted and expected paths in the process map which were not followed in real-life, highlighting implementation issues and other discrepancies.

process models. Generally speaking, we are interested to extract two types of behavior:

1. Real-life behavior which is rejected by the process model: *model correctness perspective.*
   By validating the current process model on historic process instances, deviating cases containing behavior that is explicitly disallowed by the model are detected, so that problematic or suspicious process instances can be identified and audited if necessary. However, it is possible that not all deviating instances should per definition be rejected, as it might be the case that some deviations are presenting allowed behavior after all, giving rise to process model enhancements in the form of the removal of restrictions or addition of relaxations. That is, instead of rejecting some deviating process cases which do not conform to a given model, it is also possible to modify the model to better represent the current real-life behavior. This type of behavior is thus applied to assess whether the given model is *correct* in accordance with observed, real-life data. Note that this perspective is related to the fitness quality dimension.
2. Allowed behavior which is absent in real-life instances: *model completeness perspective.*
   Additional to investigating if the given process model supports the execution of process instances as they occurred in real-life, one can also check whether all allowed behavior by a process model can also be found in an event log, or, put differently, if the process instances as they happen in real-life never exhibit certain behavior, although this behavior is nevertheless allowed by a given process model. This type of behavior is thus utilized to determine whether the given model is *complete* in accordance with observed, real-life data. Note that is different from the concept of log completeness as we have mentioned before, as the topic of interest here is the process model. Note that this perspective is related to the precision quality dimension.

Figure 4.2 provides a schematic overview of these types of behavior.

We will focus our attention on the aspect of enhancing process models, as the aspect of validation can be dealt with by using standard fitness-based metrics, as seen in Chapter 3. Furthermore, we will focus process models expressed in a declarative manner, as we have also seen how to deal with the problem of unobserved behavior for procedural models before. That is, by:



Area a: Allowed Superfluous Behavior: *Completeness* Issue
Area b: Normal, Allowed Behavior
Area c: Rejected Real-life Behavior: *Correctness* Issue
Area d: Rejected, Non-occurring Behavior

Figure 4.2: Schematic overview of behavior found when comparing a process model with real-life event log data. Permitted behavior is constrained by a process model, which is then compared with the behavior seen in the event log. Area “a” corresponds with behavior allowed by the process model but not evidenced by real-life data—indicating a completeness issue, whereas area “c” indicates real-life behavior which was rejected by the process model—indicating correctness issues. Area “b” corresponds with normal, allowed and seen behavior. Area “d” corresponds with rejected behavior, but which was also not observed in the event log.

1. Finding unobserved events in the form of artificial negative events representing behavior which never occurred but was nevertheless allowed by the process model, i.e. the false positive events. These cases help to uncover expected but non-occurring behavior.

2. Discovering a process model and annotating it based on unobserved events, as illustrated in the previous section. This cases help to uncover potential implementation problems.

Looking at declarative models is also interesting due to a reason following from a real-life aspect which comes into play when business processes are defined and developed in organizations. Oftentimes, two roles are involved in the management of organizational processes: domain experts, meaning business analysts or managers, and IT experts, who have the responsibility of translating business requirements to an effective implementation [120]. Two different perspectives can be identified between these complementary roles: while the IT expert often follows a procedural modeling style in order to better deal with implementation and execution aspects, the business analyst commonly applies a more declarative approach, describing business processes as a set of policies, rules and guidelines which guide the overall activity flow. When performing business process validation and enhancement tasks, the ability to correspond retrieved findings to decision-makers in a clear and comprehensible manner (e.g. in the form of a new rule), rather than a formal revision of an implemented procedural model is a significant advantage. As such, we will deal with these concerns by retrieving behavior which is allowed by the set of given rules but not found in a given event log, found by matching rule-generated forbidden activity executions with absent behavior in the event log.

4.3.2 Discovering Superfluous Modeled Behavior

Checking the *correctness perspective* on declarative models (i.e. the fitness) is well described in literature, i.e. in the form of rule based property verification [128]. The enhancement of declarative process models as we described above, however, has received less attention. Two reasons can be stated to provide an explanation. First, it follows from the nature of declarative models that over-specification is often (and willingly so) avoided, allowing to defer the exact control-flow sequence of activities until run-time. Second, computing all possible allowed paths from a given process model (either procedural or declarative) becomes increasingly more difficult when the size or complexity of the described process increases. Although we agree with the sentiment that the ability to generalize is one of the key benefits of declarative (process) models compared to procedural representations, we argue that investigating allowed behavior which never occurs in real-life is nevertheless valuable, for example to discover business rules which were never explicitly formulated, to discover behavior which is allowed by business and domain experts but which was not permitted in the running information system due to implementation distortions, or simply to gain insight in real-life processes. To deal with the second remark (computational difficulty), we apply the artificial negative event induction method as described above in order to introduce negative examples in historic process instances (non-occurring behavior), which are then compared with the set of negative events generated from the current rule base (model-rejected behavior). We then apply a rule induction technique in order to derive new candidate-rules.

The intricacies of this step are as follows. First, since we are interested in constraining rules which restrict the possible flow between activities, we artificially induce negative events in historic process instances to capture which behavior did not occur during the actual, real-life execution of the process at hand (non-occurring, log-rejected behavior), as was described in Chapter 3. Next, a second set of negative events is generated from the rule base (representing the disallowed, model-rejected behavior). To do so, we iterate over all events of all given traces and check which activity types other than the actually completed event could have occurred as well. Algorithm 4.1 describes this generation process. Note that we have already described a similar generation process to extract disallowed, model-rejected behavior when we induced artificial negative events from a given procedural model, see Algorithm 3.2—the basic idea here is the same, with the exception that no replay is performed on a Petri net to see whether a "negative trace" is truly disallowed or not, but instead the candidate negative event is compared with the rule base.

Once the two sets (log-rejected and model-rejected) of negative events are constructed, a comparison operation is performed in order to find those negative events which are not yet imposed by one or multiple rules (unconstrained, non-occurring behavior)[2].

[2]The actual comparison itself is exact and trivial, since the sets of negative events are inserted at the same positions in the same traces, so that comparison can be performed by iterating over each position in each event log trace, i.e. $NE(\sigma_i) \setminus NED(\sigma_i)$.

Algorithm 4.1 Rule-based generation of negative events.

```
Input: An event log $L$
Input: $\mathcal{R}$ % Given declarative process model (rule base)
Output: A set $N$ of induced artificial negative events
 1: function NED($\sigma_i$) % Induce set of negative events from declarative model
 2:     $N := \emptyset$
 3:     for all $a \in A_L \setminus \{\sigma_i\}$ do
 4:         $s := 0$ % Score for this negative event
 5:         $\tau := \langle \sigma_1, \ldots, \sigma_{i-1}, a \rangle$
 6:         for all $r \in \mathcal{R}$ do
 7:             if $r(\langle \sigma_1, \ldots, \sigma_i \rangle) \wedge \neg r(\langle \sigma_1, \ldots, \sigma_{i-1}, a \rangle)$ then
 8:                 % $r$ holds in partial original trace but not in the partial candidate negative trace
 9:                 $s := 1$ % Rule base rejects behavior, candidate negative event is a true one
10:             end if
11:         end for
12:         % Negative event with activity $a$ and weight $s$
13:         $N := N \cup \{(\sigma, i, a, s)\}$
14:     end for
15:     return $N$
16: end function
```

4.3.3 Extending the Declarative Model

Once a set of unobserved events representing superfluous behavior has been established, these events can then be analyzed in order to derive new domain knowledge. Note once more that these events describe behavior which did not occur in real-life, although the rule base does not explicitly prohibit this behavior from happening. Inspecting such cases can lead to new rules which extend the current model in order to improve its completeness, or can light up implementation distortions (something was permitted to occur by business conduct but prohibited by the implemented system).

It is not always straightforward to propose or design new rules based only on absent behavior present in an event log. However, it is possible to apply the given positive and constructed negative events towards a rule induction technique in order to automate the process extension task as follows. Based on the event log supplemented with unobserved events, a data set is constructed containing a binary target variable denoting if a particular activity of interest $a \in A_L$ occurred (positive example) or was not observed (negative example). Next, for each $\sigma_i \in \{\sigma_i \in \sigma | \sigma \in L \wedge \sigma_i = a\}$ (positive event) and for each $\sigma_i^- \in \{\sigma_i^- \in NE(\sigma_i) \setminus NED(\sigma_i) | \sigma_i \in \sigma, \sigma \in L \wedge \sigma_i^- = a\}$ (unobserved event), its history $\theta = \langle \sigma_1, \ldots, \sigma_{i-1} \rangle$ is examined and an instance is added to the data set. Each such instance contains an attribute att_a per activity type $a \in A_L$, denoting whether this activity was absent ($a \notin \theta$, attribute value equals "absent"), present ($a \in \theta$, attribute value equals "present"), or present at the final position in history θ ($a = \theta_{|\theta|}$, attribute value equals "directly precedes"). Note that other input attributes can potentially be supplemented as well (e.g. to take into account case-related attributes), but for simple control-flow based rule discovery as described here, the set of properties as described above suffices. Once the data set is constructed, well-known rule induction techniques such as RIPPER [132] can be applied in order to derive a new rule. Maruster et al. [63] also derive control-flow based rules from a given process log using a similar rule induction methodology in order to discover a process model from a given event log. Contrary to our approach, however, the authors do not take into account natural or artificial negative examples, so that the detection of non-occurring behavior becomes more troublesome. Furthermore, enhancing given declarative process models poses difficulties as well, since no comparison operation between model-rejected behavior and non-occurring behavior can be performed. Note that we target a perfect accuracy for the derivation of new rules. If misclassifications do occur, the data set can be supplemented with additional attributes which are able to perfectly distinguish between negative and positive examples in order to support the addition of a new rule to the process model. Finally, remark that we are mainly interested in deriving rules describing the occurrence of a negative event (i.e. the rejection of the execution of an activity), as the declarative model is formulated such that, by default (i.e. an empty model), all activity types are always allowed.

4.3.4 Example Case

An example is provided to illustrate the proposed approach, in the form of an insurance claim handling process. Based on the Petri net model shown in Figure 4.3, which was derived, an event log was generated containing twenty simulated process instances and 92 total events, denoting that a certain activity completed at a specific point in time within a process instance. Remark that the Petri net depicted in Figure 4.3 will not be considered further throughout the remainder of the example. The Petri net given below should thus only be used as a guidance in order to understand the possible flow of traces included in the simulated event log. The actual process model is specified in a declarative manner, and reads as follows:

Rule 1. Claim evaluation (*evaluateClaim*) must be performed at least once.

Rule 2. Approval for pay of damages (*approvePayDamages*) and the calculation of new premium (*calculateNewPremium*) are executed in parallel.

Rule 3. Claim evaluation (*evaluateClaim*) must precede a proposal for settlement (*proposeSettlement*).

Rule 4. Approval for pay of damages (*approvePayDamages*) must be preceded by claim evaluation (*evaluateClaim*).

Rule 5. Closing (*closeClaim*) and rejecting (*rejectClaim*) a claim are mutually exclusive.

We can also convert the rule base from a natural language representation to a more formal representation. Declare [133–136], SCIFF [119, 137], Linear Temporal Logic [138] and CONDEC [139] can all be applied towards this purpose. We will make use of Linear Temporal Logic (LTL) to represent the given constraints. LTL has a number of interesting advantages, the first being that it allows for an unambiguous interpretation of business rules. Secondly, many LTL "patterns" already exist in literature which are able to capture the multitude of given rules. For a detailed description of LTL semantics, we refer to [138]. The rule base is thus now represented as follows:

Rule 1. $\diamond(evaluateClaim)$

Rule 2. $\diamond(approvePayDamages) \iff \diamond(calculateNewPremium)$

Rule 3. $(evaluateClaim)W(\neg(proposeSettlement))$

Rule 4. $(evaluateClaim)W(\neg(approvePayDamages))$

Rule 5. $\neg(\diamond(closeClaim) \wedge \diamond(rejectClaim))$

Next, we discover superfluous allowed behavior by executing the steps as described above. First, we generate artificial negative events using the technique described in Chapter 3 to retrieve log-rejected behavior. As an example, consider an event execution trace from the example as given in Figure 4.3. After using the artificial negative event induction procedure, sets of negative events are inserted before the occurrence of each normal—positive—event, shown in Table 4.1 below each positive event for a single example trace. For example, after performing the activities of *claimIntake* and *reviewPolicy*, the real-data as it happened and was recorded in the event log rejects the occurrence of *claimIntake* (third column, i.e. before the occurrence of *evaluateClaim* in this trace). It is important to remark once more that the negative events are generated based on the behavior seen in real-life, i.e. as recorded in a running system.

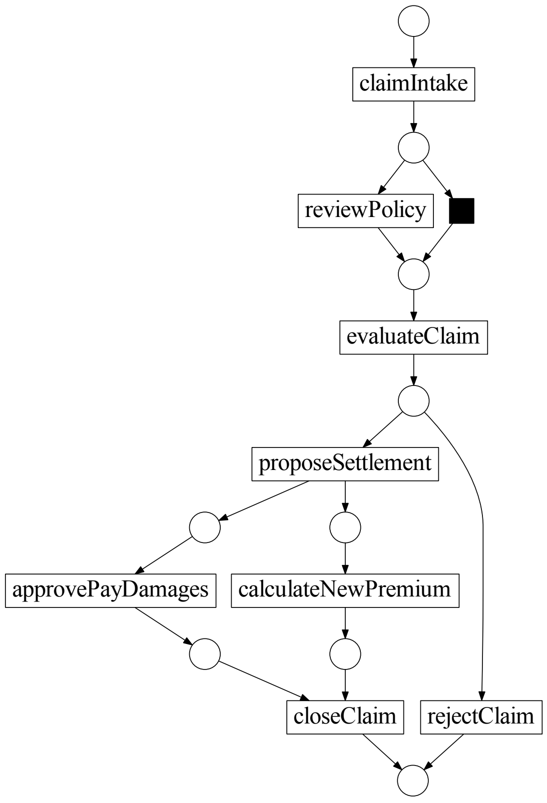


Figure 4.3: Insurance claim handling process model used to generate example event log.

Table 4.1: An insurance claim handling process instance from the example supplemented with artificially generated negative events.

$\sigma_i \in \sigma$	claimIntake	reviewPolicy	evaluateClaim	...	closeClaim
$NE(\sigma_i)$	reviewPolicy	claimIntake	claimIntake		claimIntake
	evaluateClaim	proposeSettlement	reviewPolicy		reviewPolicy
	proposeSettlement	approvePayDamages	proposeSettlement		evaluateClaim
	approvePayDamages	calculateNewPremium	approvePayDamages	...	proposeSettlement
	calculateNewPremium	closeClaim	calculateNewPremium		approvePayDamages
	closeClaim	rejectClaim	closeClaim		calculateNewPremium
	rejectClaim		rejectClaim		rejectClaim

Next, we compare this set of negative examples with the set of negative events as imposed by the current rule base (model-rejected behavior, Algorithm 4.1) to find a valid set of unobserved events. In our insurance claim handling example, the current rule base leads to the induction of 102 model-rejected negative events, denoting that a certain activity could not occur. For example, Rule 3, "Claim evaluation (*evaluateClaim*) must precede a proposal for settlement (*proposeSettlement*)", introduces negative events with activity type *proposeSettlement* at every position of each trace as long as *evaluateClaim* was not completed at a previous position. These negative events are then removed from the log-rejected 593 negative events which were generated based on historic, logged information to derive unconstrained, non-occurring behavior.

Following this, the remaining set of negative events can then be analyzed in order to derive new domain knowledge. For instance, after inspecting the set of negative events which were not captured by one or multiple rules, it is found that the activity *reviewPolicy* is frequently rejected from completing based on actual logged behavior, although no single rule in the current process model constrains this activity in any way. Therefore, we are interested in deriving a new rule which does describe when a policy may be reviewed to cover this unobserved behavior.

Based on the steps above, the following rule for the *reviewPolicy* activity type is derived after applying RIPPER (depicted in conjunctive normal form):

Rule 6. IF (NOT claimIntake = "directly precedes")
THEN reviewPolicy = 0

Or, stated otherwise: the *reviewPolicy* activity is rejected from being executed (value 0 corresponds with a negative event) if it is not directly preceded by a *claimIntake* activity. Domain experts can then decide if the rule "Reviewing insurance policy is optional but may only occur right after a new claim intake" should be added to the process model. The corresponding LTL representation for this rule would then read as:

Rule 6. $\Box(\bigcirc(reviewPolicy) \implies claimIntake) \vee \Box(\neg(reviewPolicy))$

Similarly, the *rejectClaim* activity type is also never executed at certain positions, even after taking the constraints imposed by the current rule base into account, so that the process model (Rule 5 in particular) can be made more strict. The following rules are derived and replace Rule 5:

Rule 5a. IF (evaluateClaim = “absent”)
THEN rejectClaim = 0

Rule 5b. IF (proposeSettlement = “present”)
THEN rejectClaim = 0

Our technique also deals with parallelism (AND-splits) in a correct manner. For example, the following rules are derived for *approvePayDamages*:

Rule 8a. IF (NOT proposeSettlement = “directly precedes”)
AND (NOT calculateNewPremium = “directly precedes”)
THEN approvePayDamages = 0

Rule 8b. IF (NOT proposeSettlement = “directly precedes”)
AND (approvePayDamages = “present”)
THEN approvePayDamages = 0

By following the technique as described above, the completeness of declarative process models can be improved in an automated manner by deriving new business rules based on the actual behavior of process instances. Note that this metholodogy can be extended so that declarative models can also be enhanced on the basis of information other than control-flow constructs (i.e. “presence”, “absence” and “precedence” information in the event history of process instances), so that the full flexibility of declarative process models can be leveraged. For example, by taking organizational and case data into account, more granular and refined business rules can be derived, revealing knowledge about the way people work together or how processes develop according to their specifics.

4.4 Towards an Event Existence Classification Framework

We close this chapter—and this part—with some thoughts on the development of a complete event existence classification framework. With all the discussion around positive events, negative events, artificial negative events and unobserved events given so far, we wish to emphasize that common process management and analytics techniques usually deal with one specific type (or kind) of business events only: these which are recorded in the event log (or database) under consideration and which (or so they should, that is) correspond to some real-life activity—i.e. the pure, positive events. That is, many techniques assume that the provided event logs contain complete (i.e. all possible) and accurate (i.e. compliant and absent of noise) information on the process. While the presence of a systematic, reliable and trustworthy recording of the events is essential for the theoretical development process discovery and reconstruction, in practice, inaccurate forms of event log data may be present (or desired information may be absent). The assumptions described thus lead to important implications for business applications. Firstly, the completeness assumption for the event log implies that process analyses might be conducted on a subset of the allowed process behavior [140]. Manual operations activities that are not supported by information systems might be hard to record. Consequently, the absence of a certain type of activity in the event log might not correspond with the reality. Additionally, the event log may not cover all allowable transition paths, as these transitions did not occur within the event log's timeframe. These are important implications when reviewing or reengineering a business process based on retrieved process models. Secondly, not all events present in the event log might be an honest representation of the activities performed. For example, manual activities could be performed and recorded in a different sequence or might not have happened at all. Performing a process audit based on this information may result in distorted conclusions and recommendations, e.g. a missing event might indicate fraud or be the direct result of ineffectiveness in the event recording. The research towards the defining and categorization of events has not yet been related to these general data quality aspects. Ontology-oriented contributions like DOLCE [141] and UFO [142–144] distinguish between atomic events (i.e. achievements) and complex events (i.e. accomplishments). REA [145] represents two kinds of events: economic events that represent changes in scarce means and business events that correspond to activities that the management wants to plan, monitor and evaluate. Multiple process modeling approaches incorporate events as something of importance that happen in a process model, see e.g. EPC (Event-driven Process Chain) [146]. They usually distinguish at least events preceding an activity (i.e. precondition events) and events succeeding an activity (i.e. post condition events). Many authors also propose ontologies and definitions around the notion of events in the context of the field of complex event processing [147–151]. However, our work cannot be directly related to this domain, as we approach the concept of an "event" from a different semantic perspective, namely its meaning and its types within a business processes analytics and process mining environment, that is, events as they occur in information system logs. More closely related to our context are thus the contributions in the context of data quality, with dimensions such as data accuracy, data completeness and data security [152].

In short, based on all the techniques outlined in this part of the dissertation, we see that a more complete framework is needed in order to distinguish between all the various sorts of business events that are important in the context of business process analytics. As such, we argue for the construction of a comprehensive business event analysis classification framework, with the following goals and objectives:

1. Raise awareness about the various types of events that can exist in a business context, starting from the premise that the assumption that reality implies registration within this context regarding business events is an incomplete one.
2. Providing a clear and unambiguous naming scheme when talking about "events".
3. Enabling a better orientation of these research areas and enables to indicate potential issues in the research fields.

Previous research in event-based information systems and process analysis has mainly focused on the behavior described by the registered events. Process analysis contributions have not only uncovered potentially harmful deviation between the designed process and the real process behavior, but also between the real process behavior and the event log (considering for example the notion of "invisible tasks," which are not logged in audit trails but impact the way workers "move" through processes). Based on previous works and the techniques presented thus for, we are able to propose five boolean criteria to be considered in an event analysis classification framework:

1. Occurrence in real-life. A real business event is an occurrence that happened in the organization's business environment and that is deemed relevant for the organization. The opposite is: something that did not happen.

2. Recorded event (in the event log). A recorded event is an occurrence that was registered by an information system of the organization in the event log under analysis, irrespective of the fact that it is an actual business event or not. Process analytics researchers and management scientists often assume the completeness of the event log. Concretely, this means that an assumption is made that all possible behavior is present in the event log and that the execution of all events (and activities) will be recorded in a precise and exact manner. In real world projects, these assumptions are easily challenged. For example, a manual task or a phone call will not necessarily be registered, or rare behavior will not be observed in the historic data repository under consideration, thus perhaps leading to analysts deriving that such behavior should occur (closed world assumption). Additionally, the event log may also reflect process behavior that deviates from reality, e.g. registration of non-executed activities or event records containing incorrect data due to anti-dating or employees using the authentication combination of other employees. Whether actual business events are registered may indicate whether an event log complies with reality. Secondly, while process-aware information systems may not be able to register every event relevant to the process under review, it might nevertheless be possible to find evidence in other sources, such as event logs of related process, mail history, RFID registration systems, etc.

   The combinations of these two criteria leads to four primary event type "categories", see Table 4.2.

3. Recorded event (in alternative source). A recorded event extracted from an alternative source is an occurrence that was registered by a system in a different data source, potentially but not exclusively another event log.

4. Compliant event. The classification should take into account whether an event corresponds to the allowed or permitted mode of operation or whether it is deviating from this mode. An event is compliant with the business rules if—given the existing process history—the occurrence of the event does not violate any business rule.

5. Expected event. An event is expected when its presence is foreseen and anticipated during normal business operation, whereas its absence might

Table 4.2: The combination of the "actual business event" and "recorded event" criteria leads to the derivation of four primary event type categories.

Occurred in real-life?	Recorded in event log?	Event Category
Yes	Yes	Truthful Event
	No	Invisible Event
No	Yes	False Event
	No	Unobserved Event

result in a costly, deviating or special handling of the process instance.

Based on the previously presented classification criteria we are able to discern different event types, some of which have not yet been recognized in the process analysis literature. We will not discuss all of these in depth in this work, but instead refer to literature where a detailed description of the proposed framework can be found [153]. Note however the positioning of positive, negative and unobserved events in this framework:

- Positive events as discussed so far corresponds to truthful events in Table 4.2. That is, events which occurred in real-life and were recorded in the event log. Checking these events on a process model then can establish whether they were compliant or not.
- Natural negative events can now correspond to multiple categories in the proposed framework. When the activity was actually executed in real-life, it is either recorded (truthful event) or not (invisible event), but non-compliant according to the prescriptive model. When the activity was not executed, it is still possible to log a truthful "warning" event, in line with compliance measure, as the actual negative behavior was successfully prevented.
- Artificial negative events now corresponds to unobserved events in the proposed framework. They have been utilized as such in this chapter.

This chapter concludes the first part of the dissertation and our discussion on artificial negative events. Next part will discuss a number of assorted topics and contributions not directly related to the concept of negative events.

Part II

Other Advances in Process Mining

Chapter 5

Fodina: Robust and Flexible Heuristic Process Discovery

"The battlefield is a scene of constant chaos.
The winner will be the one who controls that chaos,
both his own and the enemies."
– Napoleon Bonaparte

5.1 Introduction

In this chapter, we present *Fodina*, a process discovery technique with a strong focus on robustness and flexibility. The contribution of this work is not to propose yet another process discovery tool, but rather to pragmatically improve upon a class of existing process discovery algorithms, namely the so-called "heuristic" miners [44], adding some particular interesting features to make the approach more robust to noisy data, the ability to discover duplicate activities, and flexible configuration options to drive the discovery according to end user input. "Heuristics Miner" is one of the best known and most used process discovery algorithms both by practitioners and researchers [154], and has also proven its worth in benchmarking studies illustrating the technique's ability to discover high-quality models [47]. However, some problematic issues can be identified which negatively impact the reliability of the technique. As such, we have set out to perform a thorough review of the existing Heuristics Miner and all its variants to identify a list of issues. Then, based on a literature study regarding dependency-based heuristic process discovery techniques, we consequently propose a new implementation of a heuristic process miner which is proven to be more robust via a series of empirical experiments.

5.2 Preliminaries

5.2.1 Literature Overview

In the area of process discovery, the *α-algorithm* can be regarded as one of the most fundamental and substantial techniques (see also the introductory chapter). Van der Aalst et al. [55] prove that the technique is able to learn an important class of workflow nets (structured workflow nets) from event logs, provided that the given event log is sufficiently complete (meaning in this case that each binary sequence between two activities which can occur is also present in the event log) and that the event log does not contain any noise. Therefore, although the α-algorithm is a formal and elegant approach towards constructing process models from activity sequences, the technique is sensitive to noise and incompleteness of event logs. Furthermore, the original version of the technique is unable to discover short loops or non-local, non-free choice constructs. Other scholars have improved the α-algorithm to mine short loops [58] and detect non-free choice constructs [68] (denoted as "α++").

In order to deal with the problem of noise and to remedy the robustness problem of the α-algorithm class of techniques, Weijters et al. developed *Heuristics Miner* [44] (the approach was initially developed under the name "Little Thumb" [57], one year after the α-algorithm). This technique extends the α-algorithm in that it applies frequency information with regard to relationships between activities in an event log. Heuristics Miner can also discover short loops and non-local dependencies. Invisible activities are not mined directly as such, but the mined "Heuristics net" does not specifically require the presence of invisible activities to model activity skips or complex routing constructs (after conversion to a Petri net, the model will then contain the invisible activities necessary to represent these constructs). Duplicate activities are also not mined. Note that other process discovery techniques also make use of Heuristics nets to represent mined process models, most notably Genetic Miner [65, 66, 155, 156], although this technique is not regarded as a typical heuristic process discovery algorithm in the context of this chapter as it applies an evolutionary optimization strategy to derive a fitting process model, rather than applying frequency-based dependency measures.

Following closely after the inception of Heuristics Miner, Günther and van der Aalst developed another heuristic process discovery technique, named *Fuzzy Miner* [67], geared specifically towards mining unstructured, spaghetti like processes. Like Heuristics Miner, this technique does not mine duplicate or invisible activities. Additionally, users can apply a number of filters to create higher-level abstractions of complex process models, either by filtering out activity nodes or edges, or by clustering groups of activities in the Fuzzy model. A variant of Fuzzy Miner is implemented in the proprietary tool Disco [90], released in 2012.

In 2010, Burattin and Sperduti proposed an adaptation of the Heuristics Miner algorithm—*Heuristics Miner++*—which extends the former by considering activities with time intervals, i.e. having a starting and ending time instead of being logged as an atomic, zero-duration event [84]. The authors modify Heuristics Miner accordingly in order to better derive parallelism relations between activities based on this timing information (overlapping activities in time infer parallelism). The same representational process model form as utilized by Heuristics Miner is retained, i.e. Heuristics nets. No duplicate activities are mined. The same authors have also proposed a modified Heuristics Miner which is able to deal with streaming event data [87], and an initial strategy towards automatically optimizing the miner's parameters has been proposed as well [85].

Finally, Weijters and Ribeiro have created a modified version of their Heuristics Miner algorithm, denoted as *Flexible Heuristics Miner* [86], which outputs the mined model as a Causal net [4]. Although this representation is very similar to Heuristics nets at first appearance, an important difference exists in the way input and output bindings are expressed for each task (a binding is a set of sets of tasks—detailed preliminaries and definitions are provided below). In Heuristics nets, the subsets in a binding are in an AND relation, whereas the tasks within these subsets are interpreted as being in a XOR relation. For Causal nets, the semantics are reversed, so that the subsets of tasks in a binding are in an XOR relation, whereas the tasks within each subset are in an AND relation. The latter representation is more structured and easier to understand. The Flexible Heuristics Miner also incorporates a new strategy to mine split/join semantics, based on the frequencies each input and output binding is activated for each task. Although Causal nets (or: "C-nets") have gained traction in recent years within the process mining community due to the fact that they provide a better representational bias for process discovery compared with conventional, more strict design oriented languages (such as Petri nets or BPMN) [43], the Flexible Heuristics Miner currently remains the only process discovery technique capable of directly mining such models.

Table 5.1 provides a concluding synopsis of the various heuristic process discovery algorithms together with their various implementations. All of the heuristic process discovery algorithms described above have been implemented in various forms and stages, most commonly as a ProM plugin.

Since the implemented algorithms and techniques do not always correspond fully with the specifications outlined in the literature, it is useful to provide a global overview highlighting main versions with their properties; we only consider ProM versions from 5.2 onwards. Some discovery algorithms are also implemented in earlier version, but these are generally considered as being outdated and are no longer available. The α-algorithm is available in both ProM 5.2 and ProM 6[1]. ProM 5.2 includes three variants of the α-algorithm: Alpha Miner, Tsinghua Alpha Miner and Alpha Miner++ , whereas the latter only contains the basic variant. Heuristics Miner was originally conceived as a stand-alone implementation ("Little Thumb") but was later implemented in ProM. ProM 5.2 contains configuration options for various thresholds. A conversion plugin to convert Heuristics net to a reduced Petri net is also available. The ProM 6 implementation drops configuration options for the "positive observations", "dependency divisor", and "AND split/join" thresholds. This implementation also offers a conversion plugin to convert Heuristics net to Petri net, although the conversion is performed in a faulty manner for some cases (see also identified issues in Section 5.3 below). Note that this implementation cannot be called directly by end-users. The Flexible Heuristics Miner can be called from the plugin list in ProM 6 ("Mine for a Heuristics Net using Heuristics Miner"), but note, however, that this implementation first executes the (non-flexible) Heuristics Miner algorithm—cfr. above—and thus derives split/join semantics in this manner, i.e. using the same threshold based approach as in the (non-flexible) Heuristics Miner technique. The resulting Heuristics net is then "annotated" with frequencies corresponding to the number of times each input and output bindings (denoted as a "pattern") were activated for each task. In addition, ProM 6 also offers a "Mine for a Causal Net using Heuristics Miner" plugin, which does apply Flexible Heuristics Miner, and returns a Causal net instead of a Heuristics net. Contrary to the

[1]ProM 5.2 was the latest ProM release within the version 5 branch. At the time of writing, the latest ProM 6 release is ProM 6.3, which will be referred to as "ProM 6" throughout this chapter.

Table 5.1: Overview of heuristic process discovery algorithms.

Year	Name	Author(s)	Implemented In	Details
2003	α-algorithm	van der Aalst et al. [55]	ProM 5.2, ProM 6 (latest: 6.3)	No frequency-based heuristics to detect noise, included as inspiration behind Heuristics Miner. Extended to mine short loops [58] and to mine non-free choice constructs [68].
2003	Heuristics Miner (originally: Little Thumb)	Weijters et al. [44, 57]	Little Thumb (stand-alone), ProM 5.2, ProM 6	Applies dependency measures on frequency-based counts in order to derive a Heuristics net robust to noise.
2007	Fuzzy Miner	Günther and van der Aalst [67]	ProM 5.2, ProM 6, Disco	Geared towards discovering unstructured processes using various dependency measures, filters and abstractions.
2010	Heuristics Miner++	Burattin and Sperduti [84]	ProM 5.2	Modification of Heuristics Miner which takes into account activities' time duration.
2010	Flexible Heuristics Miner	Weijters and Ribeiro [86]	ProM 6	Revised version of Heuristics Miner using another representational language (Causal nets) and a different strategy to mine split/join semantics.
2012	Stream-aware Heuristics Miner	Burattin and Sperduti [87]	ProM 6	Modification of Heuristics Miner for discovering models from event streams.
2014	Fodina	vanden Broucke et al.	ProM 6	Proposed technique: user configurable (flexible), robust and able to discover duplicate activities.

previous plugin, the new flexible method of discovering split/join semantics is applied, both for the discovery and annotating the given Causal net with frequency statistics. Concerning Heuristics Miner++, Burattin and Sperduti offer a ProM 5.2 plugin[2] based on the Heuristics Miner implementation, taking into account activity durations. The plugin expects that all activities contain "start" and "complete" events. In case of zero duration, the behavior is equal to that of Heuristics Miner. For the Stream-aware Heuristics Miner, ProM 6 plugin source code can be obtained from the official ProM Subversion repository[3]. This implementation utilizes the ProM 6 (non-flexible) Heuristics Miner implementation in combination with pre-processed streaming log windows (sliding window or periodic interval). The novelty of this discovery technique lies thus in the preprocessing of the input offered to the miner, rather than the actual discovery step (which remains unmodified). The Fuzzy Miner, finally, has its main implementation built in ProM 5.2, although a direct port of the plugin is also available in ProM 6. A proprietary implementation is also available in the form of the well-known Disco tool[4], which does not, however, offer activity clustering or a detailed configuration of heuristic metrics to mine the fuzzy model for the sake of user-friendliness.

[2]See: `http://www.processmining.it/sw/hmpp`.
[3]See: `https://svn.win.tue.nl/trac/prom/browser/Packages/Stream/`.
[4]See: `http://www.fluxicon.com/disco`.

5.2.2 Definitions

Throughout this chapter, we will apply the definitions of an event log, Causal net and Petri net as provided in the preliminaries in the introductory chapter.

We (re)define a mapping function μ between a Causal net and event log as follows.

Definition 5.1. Task-activity mapping. Let $\mu : T_C \rightarrow A_L$ be a function mapping transitions to a label contained in A_L (the log alphabet of event log L). A similar mapping was already defined in the introduction for Petri nets and is overloaded here. Each label contained in A_L can hence be associated with one or more transitions. It follows that μ assigns a label to each task in a Causal net. Tasks in a Causal net cannot be invisible, but multiple tasks can be mapped to the same activity, i.e. duplicate tasks. $\square$

We describe in addition how a Causal net can be converted to a Workflow net.

Definition 5.2. Causal net to Workflow net conversion. Let $C_N = (T_C, t_s, t_e, I, O)$ represent a Causal net; let (T_C, D) be the dependency graph defined over C_N with $D = \{(a, b) | a \in T_C \wedge b \in T_C \wedge (a \in \square b \vee b \in a\square)\}$. $N = (P, T, F, p_i, p_o, M_0, M)$ represents the corresponding Workflow net so that:

– $P = \{p^I_a | a \in T_C\} \cup \{p^O_a | a \in T_C\} \cup \{p^D_{(a_1,a_2)} | (a_1, a_2) \in D\}$—input and output places are created for each task in the Causal net. A place is created for each arc in the dependency graph.

– $T = \{t_a | a \in T_C\} \cup T^I \cup T^O$ with $T^I = \{a^I_X | a \in T_C \wedge X \in I(a)\}$ and $T^O = \{a^O_X | a \in T_C \wedge X \in O(a)\}$—a transition is created for each task, additionally, for each input binding for each task, an unlabeled transition $\in T^I$ is created; similarly, for each output binding, an unlabeled transition $\in T^O$ is created.

– $F = \{(p^I_a, t_a) | a \in T_C\} \cup \{(t_a, p^O_a) | a \in T_C\} \cup \{(a^I_X, p^I_a) | a^I_X \in T^I\} \cup \{(p^O_a, a^O_X) | a^O_X \in T^O\} \cup \{(p^D_{(a_!,a_2)}, a^I_X) | a^I_X \in T^I \wedge a_1 \in X\} \cup \{(a^O_X, p^D_{(a_!,a_2)}) | a^O_X \in T^O \wedge a_2 \in X\}$—each labeled transition is connected to its created input and output place. The input and output places are connected with the created unlabeled transitions representing the input and output bindings. Finally, the unlabeled transitions are connected with the places representing the arcs in the dependency graph. After converting a Causal net to a Workflow net, a number of redundant invisible transitions may exist which can easily be removed in an after-step to simplify the Workflow net, preserving soundness of the Worflow net [38]. $\square$

Note that Workflow nets are not as expressive as Causal nets. Therefore, the constructed Workflow net over-approximates the behavior of the original Causal net, meaning that the Workflow net may enable a firing sequence of transitions which does not correspond with a valid firing sequence in the original Causal net [43].

5.2.3 Heuristic Dependency Based Process Discovery

This section outlines the (core) workings of existing heuristic dependency-based process discovery algorithms (such as Heuristics Miner [44] and Flexible Heuristics Miner [86]). Put broadly, the steps of these discovery algorithms all follow the same basic procedure:

1. Derive counts of "basic relations" between activities in the event log.
2. Construct a dependency graph using "dependency measures" or "causal metrics", describing the basic causal semantics between activities (follows and precedes relations).
3. Mine the semantic information, i.e. the sets of input and output bindings per activity (representing the XOR and AND splits and joins).
4. Mine long-distance dependencies (optional).

These steps are described in more detail below.

5.2.3.1 Step 1: Derive Basic Relation Counts

To perform the first step, a dependency/frequency table is constructed containing the count of basic relations found in the given event log. Following information can trivially be abstracted from the event log (assume a and b are activities $\in A_L$):

- $|a|$—the number of times activity a appears in the event log (the frequency of a), i.e. the number of times an activity σ_i occurred in the event log so that $\sigma_i = a$.

- $|a > b|$—the direct succession count between a and b (the number of times that a is directly followed by b), i.e. the number of times an activity σ_i occurred in the event log so that $\sigma_i = a \wedge \sigma_{i+1} = b$.
- $|a >> b|$—the repetition count between a and b (the number of times that a is directly followed by b and b again followed by a), i.e. the number of times an activity σ_i occurred in the event log so that $\sigma_i = a \wedge \sigma_{i+1} = b \wedge \sigma_{i+2} = a$.
- $|a >>> b|$—the indirect succession count between a and b (the number of times that a is followed by b, *but before the next appearance of* a *or* b), i.e. the number of times an activity σ_i occurred in the event log so that $\sigma_i = a \wedge \exists j > i : (\sigma_j = b \wedge \nexists j > k > i : (\sigma_k = a \vee \sigma_k = b))$. Note that every direct succession is also counted towards the indirect succession count.

5.2.3.2 Step 2: Construct Dependency Graph

Next, a dependency graph is constructed using dependency measures. Various such dependency measures have been proposed in the literature to determine the amount of "causality" between two activities. Based on user-defined thresholds, a dependency is added in the dependency graph between two activities when a dependency measure exceeds this threshold. Note that, in case of a perfect situation where the event log is completely free of noise, every basic relation occurring in the event log would be informative. The α-algorithm does just that, i.e. when $|a > b| > 0 \wedge |b > a| = 0$ there is dependency between a and b; when $|a > b| = 0 \wedge |b > a| = 0$, there is no dependency (exclusive choice) and when $|a > b| > 0 \wedge |b > a| > 0$, a and b occur in parallel. For noisy logs, however, more reliable measures are needed. Various such measures have been proposed in the literature, either in the context of a heuristic dependency-based process discovery algorithm, or in related work where the concept of activity dependencies is also utilized, e.g. in [63], where such metrics are used as the inputs to construct a data set, which is subsequently used in a rule learning task to derive rules explaining the relation between tasks. We provide an overview of possible dependency measures below.

Weijters et al. [44, 86] use the following dependency measure in the Heuristics Miner and Flexible Heuristics Miner to assess the level of causality between two activities a and $b \in A_L$:

$$\frac{|a > b| - |b > a|}{|a > b| + |b > a| + d}$$

This dependency measure takes the occurrence of activity a after b as counter-evidence that b is dependent on a. Even when b is never followed by a, the metric will never completely reach a value of 1 so long as $d > 0$, denoting that absolute confidence in a dependency relation is never completely possible based on the given information (the default value for d in (Flexible) Heuristics Miner equals 1).

The same authors have proposed another causality metric which also takes into account the distance between a and b [56, 57] and is computed as follows. Every time activity a is followed by b (before the next occurrence of a), a counter C is increased with a factor δ^n, with δ a causality factor $\in [0, 1]$ and n the number of intermediate activities between a and b (the contribution to C is maximal when $n = 0$, i.e. when a and b directly follow one another). Finally, when the calculation of C is finished, the causality metric is then calculated as:

$$\frac{C}{|a|}$$

Maruster et al. also define a variety of causality metrics [63]. Their first metric is equal to the equation above, but divides the summed metric by the minimum of the overall frequency of activity a and b:

$$\frac{C}{min(|a|, |b|)}$$

To measure the tendency of succession $|a > b|$ versus $|b > a|$, the authors propose a local metric as follows:

$$\hat{p} - z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n+1}}$$

with $\hat{p} = \frac{|a>b|}{n+1}$, $n = |a > b| + |b > a|$.

The idea of this metric is borrowed from statistics and is frequently used to calculate the confidence intervals for errors ($z_{1-\alpha/2} = 1.96$ for a 95% confidence level). In this case, the difference in magnitude of $|a > b|$ versus $|b > a|$ is compared.

Note that another binomial proportion confidence interval, i.e. the Wilson score interval, can also be applied as a dependency measure, which has better properties than the normal approximation:

$$\frac{\hat{p} + \frac{z_{1-\alpha/2}^2}{2n} - z_{1-\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n+1} + \frac{z_{1-\alpha/2}^2}{4n^2}}}{1 + \frac{z_{1-\alpha/2}}{n}}$$

with $\hat{p} = \frac{|a>b|}{n+1}$, $n = |a > b| + |b > a|$.

Finally, Maruster et al. also define a "global metric" as follows:

$$(|a > b| - |b > a|)\,\frac{|L|}{|A| \times |B|}$$

Mining short loops (meaning loops of length one such as in $\langle a, b, b, b, b, c\rangle$ or length two such as in $\langle a, b, c, b, c, b, c, d\rangle$ requires the use of additional metrics, since, in this case, the counter-evidence of $|b > a|$ will outweigh $|a > b|$, making many of the metrics listed above unreliable to discover such structures. As such, Weijters et al. propose the following definitions [44, 86]:

$$\frac{|a > a|}{|a > a| + 1}$$

for length one loops, and

$$\frac{|a >> b| + |b >> a|}{|a >> b| + |b >> a| + 1}$$

for length two loops.

Putting all of the above together, the Heuristics Miner algorithm defines three thresholds: dependency threshold, length one loop threshold and length two loop threshold. Dependencies between tasks that have a dependency measure above the value of the associated threshold will be accepted in the dependency graph. Heuristics Miner also includes an "all tasks connected" heuristic which ensures that each task in the dependency graph has at least one incoming and outgoing arc (except for the start and end tasks). To do so, the best candidate (i.e. highest $|a > b|$ value) is taken to determine the primal causal and dependent task. All the dependency measures discussed above have been taken in consideration during the design of Fodina. Section 5.4 describes in detail the approach followed by our proposed miner.

5.2.3.3 Step 3: Mine the Semantic Information

The next step describes the characterization of the splits and joins in the dependency graph. In the original definition of the Heuristics Miner algorithm [44], the following metric is applied to derive confidence towards two tasks $\mathtt{b}$ and $\mathtt{c}$ occurring in parallel (both dependent on $\mathtt{a}$):

$$\frac{|\mathtt{b} > \mathtt{c}| + |\mathtt{c} > \mathtt{b}|}{|\mathtt{a} > \mathtt{b}| + |\mathtt{a} > \mathtt{c}| + 1}$$

For output bindings containing more than two tasks, an iterative approach is utilized where tasks are combined in an AND relation if the value of the metric above for these tasks combined with each current member of the relation exceeds a given threshold. We refer to the literature for a comprehensive description [44].

In [63] the decision between exclusiveness and parallelism is made simply based on the following frequencies:

$$\frac{|\mathtt{b} > \mathtt{c}|}{min(|\mathtt{b}|, |\mathtt{c}|)} \text{ and } \frac{|\mathtt{c} > \mathtt{b}|}{min(|\mathtt{b}|, |\mathtt{c}|)}$$

For exclusiveness between $\mathtt{b}$ and $\mathtt{c}$, both values should be zero or small, for parallelism, both should be relatively high.

In the Flexible Heuristics Miner algorithm, XOR and AND relations are mined in a different manner [86]. For an activity $\mathtt{a}$ with depending activities $\mathtt{b}$ and $\mathtt{c}$, counts are calculated corresponding with the number of times $\mathtt{a}$ was followed by $\mathtt{b}$ only, $\mathtt{c}$ only or by both $\mathtt{b}$ and $\mathtt{c}$. However, since both $\mathtt{b}$ and $\mathtt{c}$ might be activated by tasks other than $\mathtt{a}$, one cannot simply count the number of times $\mathtt{b}$ and $\mathtt{c}$ appear after $\mathtt{a}$ (but before the next occurrence of $\mathtt{a}$). Instead, all causal tasks before $\mathtt{b}$ and $\mathtt{c}$ should be checked (i.e. connected as an input in the dependency graph)—which includes $\mathtt{a}$—to determine whether $\mathtt{a}$ was the nearest candidate appearing before $\mathtt{b}$ and $\mathtt{c}$. Based on the frequency of the different possible "patterns" (i.e. $\{\{\mathtt{b}\},\{\mathtt{c}\}\}$ and $\{\{\mathtt{b},\mathtt{c}\}\}$), an output binding is chosen. The exact procedure on how this final decision is made is left unspecified in [86]. Recall, however, the remark made in Subsection 5.2.1 regarding the implementation of the Flexible Heuristics Miner algorithm. One available plugin in ProM 6—"Mine for a Heuristics Net using Heuristics Miner"—uses the non-flexible metrics (Equation (9)) to discover the semantics of the split and joins and annotates the nets using the pattern-based approach described above with frequency information after a Heuristic net has been discovered. The "Mine for a Causal Net using Heuristics Miner" plugin does use the pattern-based approach, but includes all discovered split and join patterns in the final causal net, making the approach less robust to noise.

5.2.3.4 Step 4 (optional): Mine Long-distance Dependencies

Finally, in a last optional step, the long-distance dependencies are mined. To do so, another dependency metric with associated threshold is defined, making use of the value of $|\mathtt{a} >>> \mathtt{b}|$ [86]:

$$\frac{|\mathtt{a} >>> \mathtt{b}|}{|\mathtt{b}| + 1}$$

However, many activity-pairs exist for which this metric will also return a high value (e.g. between the starting activity and many other activities), although no additional dependency should be added. Therefore, a check is performed to see whether it is possible to go from task $\mathtt{a}$ to the ending task in the dependency graph without having visited $\mathtt{b}$. If this is possible then the additional long-distance dependency is added to the dependency graph (as this makes the graph more precise).

The Flexible Heuristics Miner [86] uses another dependency measure to check for long-distance dependencies:

$$\left(\frac{2 \times (|\mathtt{a} >>> \mathtt{b}|)}{|\mathtt{a}| + |\mathtt{b}| + 1}\right) - \left(\frac{2 \times abs(|\mathtt{a}| - |\mathtt{b}|)}{|\mathtt{a}| + |\mathtt{b}| + 1}\right)$$

This metric adds the additional requirement that the frequency of tasks $\mathtt{a}$ and $\mathtt{b}$ should be about equal. After modifying the dependency graph, the description of the algorithm states that semantic information (AND and XOR joins and splits) should be recomputed.

This concludes the preliminary section with the overview of the Heuristics Miner algorithm and its variants, and the various dependency metrics described in the literature.

5.3 Identified Issues

The value of the existing (Flexible) Heuristics Miner algorithms should not be understated, due to its robustness to noise, ease-of-interpretation and speed. Iit

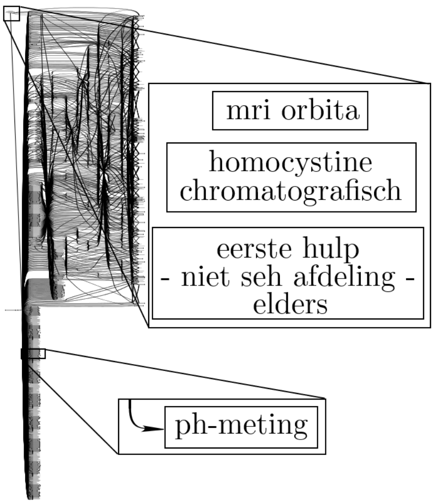

mri orbita
homocystine chromatografisch
eerste hulp - niet seh afdeling - elders
ph-meting

Figure 5.1: Some complex event logs result in process models containing unconnected tasks or tasks which are not on a path from the starting to ending task.

remains one of the most often applied and best performing process discovery algorithms [47, 154]. However, some issues can still be identified which open up opportunities for solid improvements. Some of these issues are the result of faulty or incomplete implementations, whereas others are the result of certain concepts which are somewhat vague or could be better defined. The following paragraphs provide a comprehensive overview.

5.3.1 Unconnected Tasks

Even when enabling the "all tasks connected" option, the particular complexity of several event logs (such as the "hospital log" used in the BPI 2011 Challenge[5]) causes some tasks to remain unconnected. Figure 5.1 depicts a dependency graph mined using the Heuristics Miner algorithm in ProM 6 illustrating the issue.

The issue is due both to theory and implementation. The way task connection is ensured is by adding input and output arcs corresponding with the highest dependency measure for each task, which does not ensure full connectedness in the resulting process model.

[5]DOI: doi:10.4121/d9769f3d-0ab0-4fb8-803b-0d1120ffcf54 , see `http://data.3tu.nl/repository/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54`.

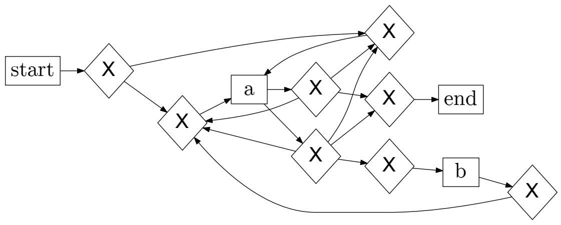


Figure 5.2: Non-fitting Heuristics net for the trace ⟨start, a, a, a, b, a, a, a, end⟩. Arcs leaving from and joining in the tasks represent an AND-split/join. The diamond shaped gateways represent XOR-splits/joins.

5.3.2 Non-fitting Process Models

One would expect that a process discovery algorithm would be able to return a perfectly fitting process model in case where the given event log only contains one single trace variant. Considering for a moment that duplicate activities could be mined, such a process model could simply model the sequence of events as they occur in the trace variant to obtain such a fitting model. The current versions of Heuristics Miner, however, are not able to do so in all cases. Figure 5.2 shows the Heuristics net obtained after mining the trace ⟨start, a, a, a, b, a, a, a, end⟩ (with all thresholds set to their lowest value). The reported Improved Continuous Semantics (ICS) fitness is 0.66 (below the maximum of 1). The Heuristics Miner-based Causal net miner present in ProM 6 shows the same issue, whereas the Heuristics Miner implementation in ProM 5.2 does mine a fitting process model. It is interesting to observe how many process discovery algorithms fail such single trace robustness test.

This issue is due both to theory and implementation. In some cases, the interrelatedness of the dependency thresholds makes it difficult to configure a parameter vector which results in a fitting model, in other cases, the inability to mine duplicate tasks make the resulting model too entangled. Finally, in some cases, split and join semantics cannot be derived in a correct manner.

5.3.3 ICS Fitness Calculation

The manner by which fitness is reported for mined Heuristics nets is currently not particularly well described. The fitness reported in the implemented Heuristics Miner is often referred to in literature as the Improved Continuous Semantics

(ICS) fitness. To be exact, the fitness metric reported is the $\texttt{PF}_{complete}$ metric as described by Alves de Medeiros [66], with traces being parsed using a continuous semantics token game, meaning that the execution of a trace is continued after the occurrence of an error (a non-fitting activity was encountered). The exact manner however how this parsing (or replay) of traces occurs in a Heuristics net is not clearly specified in literature.

In addition, there is another aspect which warrants mentioning in this context. As shown in the following code fragment[6], after calculating the ICS fitness (which is also assigned to the Heuristics net object), a final operation is performed, namely "`net.disconnectUnusedElements()`", which changes the structure of the net and can also influence the fitness of the Heuristics net, thus leading to the net having a modified fitness value. The ICS fitness is, however, not recalculated to reflect these changes:

```
//DTContinuousSemanticsFitness fitness1 =
// new DTContinuousSemanticsFitness(log);
//fitness1.calculate(population);

ContinuousSemantics fitness1 = new ContinuousSemantics(logInfo);
fitness1.calculate(population);

//Message.add("Continuous semantics fitness ="+
population[0].getFitness());
//Message.add("Continuous semantics fitness ="+
population[0].getFitness(), Message.TEST);

ImprovedContinuousSemantics fitness2 =
new ImprovedContinuousSemantics(logInfo);
fitness2.calculate(population);

//Message.add("Improved Continuous semantics fitness =" +
population[0].getFitness());
//Message.add("Improved Continuous semantics fitness =" +
population[0].getFitness(),
// Message.TEST);

net.disconnectUnusedElements();

return net;
```

This issue is due to implementation.

[6]Heuristics Miner code fragment from "`HeuristicsMiner.java`" (version obtained on 23 January 2014, revision 6737). Note the `net.disconnectUnusedElements()` operation which is performed after the fitness has been assigned to the Heuristics net.

5.3.4 Incorrect Conversion to Petri Nets

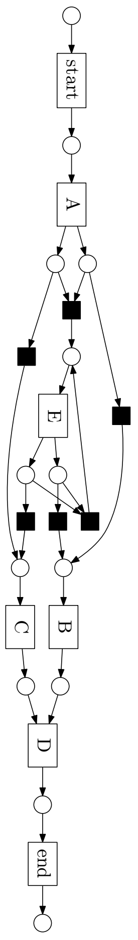
The current version of Heuristics Miner (ProM 6) contains a faulty implementation of a Heuristics net to Petri net convertor. In order to illustrate the problem, consider the Petri net process model of Figure 5.3(a), which was used to generate an event log. When mining this event log with Heuristics Miner, the output bindings of activity A are set correctly, namely as $\{\{E\},\{B,C\}\}$ (E, or B *and* C should follow A). However, when using the conversion technique to convert from Heuristics nets to Petri nets, the Petri net of Figure 5.3(b) is obtained, where now A is followed by E, or B (only), or C (only). Clearly, this is not the desired behavior. Note that this issue is not present in the Heuristics Miner version of ProM 5. The reason behind this issue is an erroneous interpretation of the semantics of the input and output bindings for Heuristics nets, which differ from those found in Causal nets. That is, within the Heuristics net, the output bindings of activity A are set to $\{\{B,E\},\{C,E\}\}$. Note once more the somewhat ambiguous semantics present when using Heuristics nets, as the bindings now represent a conjunctive set of disjunctive subsets. In addition, there is another intricacy present which disallows choosing the combination of E (first subset) and C (second subset), as selecting an activity in one subset also implies the selection of this same activity when it appears in other subsets. Clearly, the definition of the bindings as used within Causal nets is more straightforward.

In addition, the ProM 6 implementation of Heuristics Miner suffers from another issue related to Petri net conversion—albeit not directly related to the implementation of the miner itself. Contrary to the Heuristics Miner in ProM 5, the ProM 6 miner does not include the post-mining step of removing redundant invisible transitions. Instead, ProM 6 offers a separate plugin to do so. However, the "Fusion of Parallel Transitions"-Murata rule [38] is enabled by default and removes invisible transitions modeling activity "skips", thus negatively impacting the behavioral properties of the converted Petri net (the reduced Petri net will not be as fitting). ProM 5 correctly handles these cases.

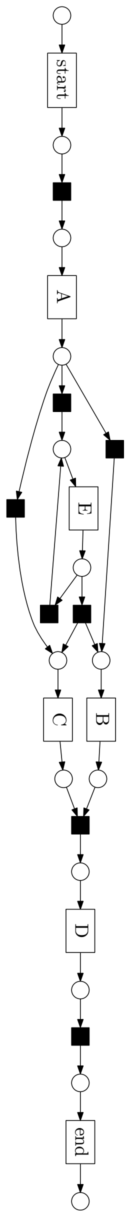
This issue is due to implementation.

5.3.5 Mining Duplicate Tasks

As mentioned above, no heuristic process discovery algorithm is able to mine duplicate activities. Alves de Medeiros et al. also use Heuristics nets as the process

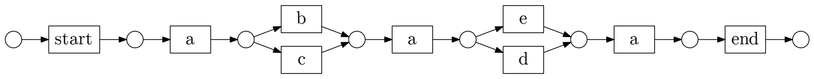


(a) Original Petri net model.

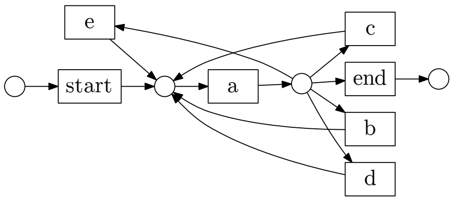


(b) Erroneous Petri net model after converting mined Heuristic net.

Figure 5.3: An original Petri net model and result after mining on generated event log and converting to Petri net with Heuristics Miner.



(a) Original Petri net model.



(b) More complex Petri net model after converting mined Heuristic net.

Figure 5.4: An original Petri net model and result after mining on generated event log and converting to Petri net with Heuristics Miner.

representational language, and apply a genetic algorithm [65, 155, 156] which is able to mine duplicate activities, where duplicate activities containing shared input or output tasks are punished. However, running this Genetic Miner suffers from a high computational complexity.

The ability to detect duplicate tasks could nevertheless greatly improve the understandability and structural clarity of the obtained process model. To illustrate why this is the case, Figure 5.4 shows yet another comparison between an original Petri net model and the (correctly converted) Petri net model after running Heuristics Miner. The fact that all activities sharing the same label are treated as a single task in the process model leads to the creation of extra arcs and dependencies and thus more structurally complex results. Even although the mined Petri net does perfectly fit the behavior in the event log, the ability to discover duplicate tasks would greatly improve the understandability and clarity of discovered process models.

This issue is due to theory, as the current theory does not describe a method to mine duplicate tasks.

5.3.6 Long-distance Dependencies

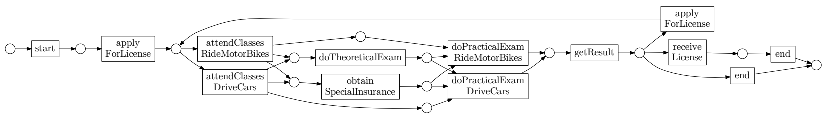
The way long-distance dependencies are constructed differs somewhat in the actual implementation of Heuristics Miner compared to the approach as described in the literature [44, 86]. The following code fragment[7] shows that the $|\mathtt{a} >>> \mathtt{b}|$ count between two tasks is only incremented once within the same trace, although multiple occurrences of the same $\mathtt{a} >>> \mathtt{b}$ pattern can exist within the same trace. For example, the trace $\langle \mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{b} \rangle$ then has $|\mathtt{a} >>> \mathtt{b}| = 1$ $(\langle \underline{\mathtt{a}, \mathtt{b}}, \mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{b} \rangle)$ while in fact $|\mathtt{a} >>> \mathtt{b}| = 2$ $(\langle \underline{\mathtt{a}, \mathtt{b}}, \mathtt{a}, \underline{\mathtt{a}, \mathtt{b}}, \mathtt{b} \rangle)$ according to the definition (recall that the indirect successions found in this trace are also considered as direct successions):

```
ArrayList<Integer> lastEvents =
new ArrayList<Integer>(trace.size());
[...]
for (XEvent event : trace) {

// XExtendedEvent extendedEvent = XExtendedEvent.wrap(event);

Integer eventIndex = null;
[...]
String eventKey = logInfo.getEventClasses().getClassOf(event).getId();
// String eventKey = eventName + "+" + eventTransition;
eventIndex = keys.get(eventKey);

if (!lastEvents.contains(eventIndex)) {

for (Integer index : lastEvents) {

// update long range matrix
metrics.incrementLongRangeSuccessionCount(index,eventIndex,1);
}
}
[...]
lastEvents.add(eventIndex);
[...]
}
```

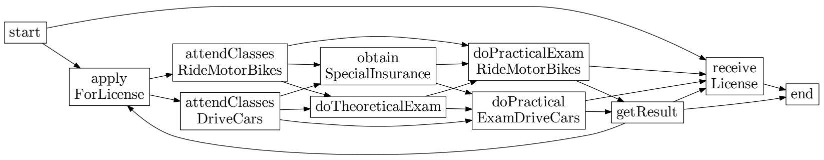
This issue is due to implementation.

Furthermore, the current long-distance dependency definition in Heuristics Miner is somewhat overly sensitive regarding. Figure 5.5 illustrates why this is the case. In Figure 5.5(a), the original Petri net is shown, containing two long-distance dependencies. Using the default long-distance dependency threshold in Heuristics Miner, these long-distance dependencies remain undiscov-

[7]Heuristics Miner code fragment from "HeuristicsMiner.java" (version obtained on 23 January 2014, revision 6737). The way $|\mathtt{a} >>> \mathtt{b}|$ is calculated differs from the approach described in literature.



(a) Original Petri net model containing two long-distance dependencies (non-local non-free choice) between *attendClassesDriveCars* and *doPracticalExamDriveCars*, and between *attendClassesRideMotorBikes* and *doPracticalExamRideMotorBikes*.



(b) Dependency graph mined with Heuristics Miner containing unwanted long-distance dependency between *start* and *receiveLicense*.

Figure 5.5: An original Petri net model and result after mining on generated event log and converting to Petri net with Heuristics Miner.

ered. Lowering the threshold allows to discover the long-distance dependencies, but also causes an unwanted long-distance dependency between `start` and `receiveLicense` to show up in the resulting process model (shown in Figure 5.5(b) as a dependency graph). A check is performed to see whether it is possible to go from `start` to the final task (`end`) in the dependency graph without visiting `receiveLicense` in order to prevent the creation of unwanted long-distance dependencies, but since this is in fact possible—meaning: `receiveLicense` is not always visited—the dependency is added. However, since `start` always occurs in each possible trace anyway, this long-distance dependency should not explicitly be modeled.

This issue is due to theory.

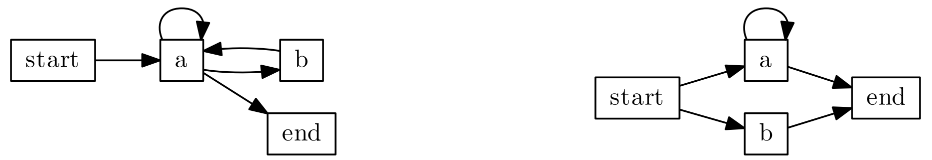
Finally, a word should be devoted to mention the fact that splits and joins (the AND and XOR relations) should be recalculated after adding long-distance dependencies in the dependency graph. This is not done as such in the implementation of Heuristics Miner, however. Instead, long-distance dependencies are just added to the existing input and output bindings such that the dependent task is added as an obligation to each subset.

5.3.7 Mining Split and Join Semantics

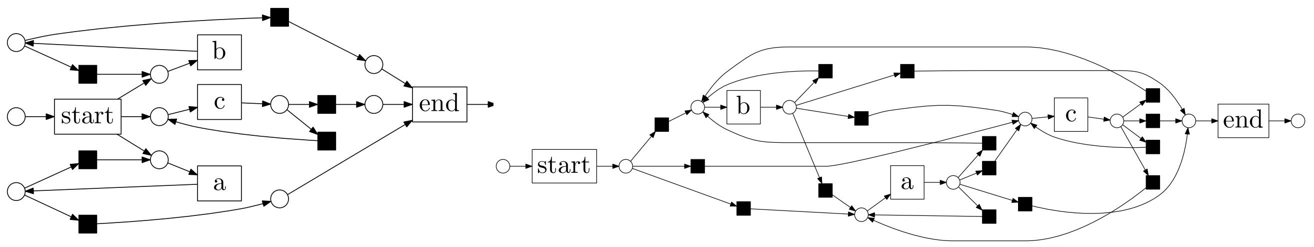
Lastly, we devote some attention to the way splits and joins are mined in the currently available set of heuristic discovery algorithms. Although the recommended method to mine the AND and XOR relations in the input and output bindings is to make use of pattern-based techniques as described in [86], one implementation of the Flexible Heuristics Miner still uses the non-flexible metric-based technique as indicated before. A second implementation, geared towards the discovery of Causal nets (as opposed to Heuristics nets) is also available, which does use pattern-based frequency counting to discover and annotate the split and join semantics, but each seen pattern is included in the resulting Causal net, which makes the implementation sensitive to noise.

To illustrate this, consider Figure 5.6 where two extreme examples are depicted. Consider first once more the event log containing only the trace $\langle \mathtt{start}, \mathtt{a}, \mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{a}, \mathtt{a}, \mathtt{a}, \mathtt{end} \rangle$. On the left hand side of Figure 5.6(a), the process model is shown after mining with Heuristics Miner. Recall from Subsection 5.3.2 that the output bindings of $\mathtt{a}$ are incorrectly mined, but let us consider for a moment these to be correct, i.e. $\mathtt{O(a)} = \{\{\mathtt{A}\},\{\mathtt{B}\},\{\mathtt{END}\}\}$. However, even when considering the correct bindings, it can still be argued that it is possible to obtain a better process model as shown in the right hand side of Figure 5.6(a), namely with an AND split from $\mathtt{start}$, as this prevents multiple executions of $\mathtt{b}$—behavior which was not seen in the given trace. Of course, without the event log being complete enough, deciding which process model is "right" is a subjective matter (perhaps just modeling the sequence as is can be regarded as an even better solution). In any case, however, it would be beneficial to offer users the possibility to communicate their preferences to this regard. As will be seen below, we have added the option to do so in Fodina. Next, the example in 5.6(b) shows two Petri nets converted (correctly) after mining for a log containing the set of traces starting with $\mathtt{start}$ and ending with $\mathtt{end}$ and with all possible permutations of $\mathtt{a}$, $\mathtt{b}$ and $\mathtt{c}$ between them with length three, i.e. $\langle \mathtt{start}, \mathtt{a}, \mathtt{a}, \mathtt{a}, \mathtt{end} \rangle$, $\langle \mathtt{start}, \mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{end} \rangle, \ldots, \langle \mathtt{start}, \mathtt{c}, \mathtt{c}, \mathtt{b}, \mathtt{end} \rangle, \langle \mathtt{start}, \mathtt{c}, \mathtt{c}, \mathtt{c}, \mathtt{end} \rangle$. The net on the left shows the process model obtained after mining with Heuristics Miner. Note that the AND and XOR relations in the $\mathtt{start}$-split are not mined correctly (output binding $\mathtt{O(start)} = \{\{\mathtt{a}, \mathtt{b}, \mathtt{c}\}\}$), since traces not containing all activity types (such as $\langle \mathtt{start}, \mathtt{a}, \mathtt{a}, \mathtt{b}, \mathtt{end} \rangle$ cannot be fittingly executed). The Petri net on the right shows the desired outcome (mined with Fodina).

This issue is due both to implementation and theory.



(a) Mined dependency graphs for the trace ⟨start, a, a, a, b, a, a, a, end⟩. On the left, a length two loop is used while on the right side, a split in start is modeled, preventing multiple executions of b.



(b) Mined Petri nets (after conversion) for the event log containing traces starting with start and ending in end with all possible permutations of a, b and c between them. The process model on the left is incorrect, since it prevents the execution of traces not containing all activities (e.g. ⟨start, a, a, b, end⟩).

Figure 5.6: Two examples to illustrate differences in process model quality.

5.4 Process Discovery with Fodina

5.4.1 Discovering Causal Nets

Based on a thorough literature study (Subsection 5.2.1), containing a breakdown of all available and currently applied heuristic process discovery variants (Subsection 5.2.3) and inspecting the issues present in these variants (Section 5.3) as outlined above, we now propose a new heuristic process discovery algorithm—named Fodina—which aims to provide a robust iteration of this set of techniques in order to mine Causal nets, including also some new features which will be discussed in the remainder of this section.

An overview of the steps performed by Fodina to mine a Causal net is given as follows (note the similarities and differences with Heuristics Miner):

1. Convert the event log to a "task log". Contextual information is used to (optionally) mine duplicates.
2. Derive counts of "basic relations" between activities in the event log.
3. Construct a dependency graph using "dependency measures" or "causal metrics", describing the basic causal semantics between activities (follows and precedes relations).
4. Set the start and end task.
5. Resolve binary conflicts (optional).
6. Assure each task is reachable in the dependency graph (optional).
7. Mine long-distance dependencies (optional).
8. Mine the semantic information, i.e. the sets of input and output bindings per activity (representing the XOR and AND splits and joins).

5.4.1.1 Step 1: Construct Task Log

In the first step, the given event log is converted to a "task log", where each activity is the event log (i.e. in A_L) is mapped to a task to-be included in the resulting Causal net (i.e. to T_C). When not mining duplicate tasks, this mapping is trivial ($\forall t \in T_C : [\exists! a \in A_L : [\mu(t) = a]]$). When the option is set to mine duplicates, the same activity in the event log can be mapped to multiple tasks in T_C. To determine which activities should be duplicated, we apply a strategy inspired by Genetic Miner [66]. In this technique, it is assumed that duplicate tasks can be distinguished based on their local context, meaning the set of input and output elements of the duplicates. The aim of Genetic Miner is then to mine process models in which duplicates of a same task do not have input or output elements in common. This approach has a couple of benefits. For example, parsing Causal nets with duplicate tasks remains relatively simple, because the output elements of the duplicates are sufficient to choose which duplicate task to fire (looking forward to the next events in the trace). In Genetic Miner, the genetic algorithm's fitness measure incorporates a punishment factor where an individual in the population is punished whenever duplicates of a same task have common input or output elements. Based on this, we have included a procedure which directly infers the duplicate tasks from the given event log, so that the time-consuming step of running a genetic algorithm is avoided. To do so, we apply the same principle of "local context" as explained above. Say that we are trying to derive if an activity a in the event log L should be duplicated. We construct a set of contexts $C = \{(\sigma_{i-1}, \sigma_i, \sigma_{i+1}) | \sigma \in L, \sigma_i = a\}$. Next, we construct the set of grouped contexts $C' = \{c \in \mathcal{P}(C) | \forall (x, y, z) \in c : [\nexists (i, j, k) \in C \backslash c : j = y \land (i = x \lor k = z)]\}$, which corresponds with the duplicate tasks to be placed in the Causal net. As an example, consider the event log: $\{\langle start, a, b, c, a, d, e, a, end\rangle, \langle start, a, c, b, a, d, e, a, end\rangle, \langle start, a, b, c, a, e, d, a, end\rangle, \langle start, a, c, b, a, e, d, a, end\rangle\}$. The set of contexts for activity a is then $\{(start, a, b), (start, a, c), (c, a, d), (b, a, d), (c, a, e), (b, a, e), (e, a, end), (d, a, end)\}$. The set of grouped contexts is constructed so that the local contexts of a are separated so that they do not overlap with one another: $\{\{(start, a, b), (start, a, c)\}, \{(c, a, d), (b, a, d), (c, a, e), (b, a, e)\}, \{(e, a, end), (d, a, end)\}\}$, representing three duplicate tasks.

Note that deriving duplicate tasks like this might indeed lead to the duplication of activities which could nevertheless be kept as a single task in the process model without impacting fitness or heavily affecting understandability. However, the choice is made to ignore this—since these "redundant" duplicate tasks still remain easy to interpret in the final process model. However, to mitigate against noise leading towards the derivation of undesired duplicate tasks (consider for example an activity which is inserted at a random position in the event log and is thus likely to be surrounded by an unseen context), we introduce a "duplicate task threshold", which works as follows. When a duplicate task is created with a frequency which is below a the duplicate task threshold ratio t (frequency of this particular duplicate task over frequency of all duplicate tasks), the duplicate task for this particular context is removed and merged with the duplicate task having the greatest frequency in the event log. Note that an alternative strategy consists of ignoring such noisy events altogether. However, we consider such "cleaning" of event logs (removing activities in the traces) as a pre-discovery task which should be executed before invoking Fodina. In addition, an option was added to better allow the duplication of activities which also repeat. Consider for example again the trace $\langle start, a, a, a, b, a, a, a, end \rangle$. Based on this, the following set of grouped contexts would be constructed for a: $\{\{(start, a, a), (a, a, a), (b, a, a), (a, a, end)\}\}$. In other words, the context (a, a, a) causes that no duplicate tasks can be found for activity a. Therefore, we allow to "collapse" repeated tasks during the derivation of duplicates, so that the two duplicate tasks for a can then be discovered (before b and after b). For the remainder of this chapter, we can thus assume the mapping function μ as a given, which is able to unambiguously map activities occurring in traces in an event log to a task occurring in the causal net. Given the way duplicate tasks are dealt with, μ is able to map an activity to one single task in the causal net event even when this activity might be duplicated in the causal net, by inspecting the local context of this activity.

5.4.1.2 Steps 2, 3 and 4: Derive Basic Relations and Construct Dependency Graph

The second step (derivation of basic relation counts) is performed completely similar as done in Heuristics Miner (see Subsection 5.2.3.1), making sure, however, to correctly derive the $|a >>> b|$ information.

To construct the dependency graph, we make use of a similar dependency measure as in Heuristics Miner, as we have found based on experiments—evaluating and analyzing the dependency measures discussed in Subsection 5.2.3—that this best reflects the user's expectations and has the best impact on the outcome of the process model. For normal dependencies (discovered first), however, we do apply a different metric, i.e:

$$\frac{|a > b|}{|a > b| + |b > a| + d}$$

(with d the dependency divisor, by default set to 1), as we argue that the direct succession of a task a after b is not always suitable direct counter-evidence against the direct succession of b after a. (The metric now also lies in the range $[0, 1]$.) For length one (discovered second) and length two (third) loops, recall the use of the metrics

$$\frac{|a > a|}{|a > a| + d}$$

and

$$\frac{|a >> b| + |b >> a|}{|a >> b| + |b >> a| + d}$$

respectively. All associated thresholds in Fodina also operate separately from each other during the construction of the dependency graph, which is not the case in the Heuristics Miner implementation, where changing one threshold (e.g. to include more dependency arcs) might have no effect without also lowering other thresholds (since multiple comparisons are performed, including against a non-configurable "positive observations" threshold for the ProM 6 implementations), which in turn might cause other undesired dependencies to show up. We have removed the "positive observations" and "relative-to-best" thresholds in the Fodina implementation, as it was observed that their impact is negligible in most cases (or covered by the other thresholds) and that most end-users do not change their default values [154].

During the discovery of length two loops to add to the dependency graph, users have the option to prohibit a length two loop dependency between a and b (from a to b and b to a) when these two tasks are both already involved in a length one loop with themselves (length one loops are mined before length two loops). This can be beneficial in cases where both activities are length one loops and both dependent in an AND relation on the same, third activity, leading to observations such as $\langle start, a, b, a, a, a, b, b, a, b, b, b, end \rangle$. This trace can then be configured to be modeled in two ways, either with a length two loop (and a XOR split/join) or without a length two loop (with an AND split/join).

In the fourth step, the start and end tasks are set in the dependency net. If desired, users can specify that artificial starting and ending tasks should be used, which are prepended and appended respectively to each trace in the event log, and connected with the other activities using the same dependency metrics as described above.

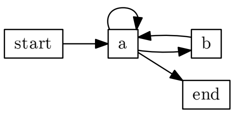
5.4.1.3 Step 5 (optional): Resolve Binary Conflicts

As stated above, during the discovery of length two loops users may prohibit a length two loop dependency when the two associated tasks are both already involved in a length one loop with themselves.

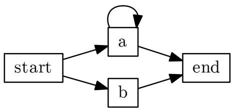
Additionally, applying the concept of "binary conflicts" as described in [67, 157] (Fuzzy Miner), users have the option to enable Fodina to try to convert *all* length two loops to a single AND relation whenever possible. As such, the trace $\langle \mathtt{start}, a, a, a, b, a, a, a, \mathtt{end} \rangle$, for example, can now be mined in three different (all fitting) ways, as depicted by Figure 5.7. The first net (a) depicts a dependency net obtained without mining for duplicates and allowing binary conflicts. Contrary to Heuristics Miner, all splits and joins are in a XOR relation, which thus leads to a fitting process model. The dependency graph in (b) resolves the binary conflicts and relinks the length two loop between a and b so that they are both dependent on $\mathtt{start}$. In the final step (mining of split and join semantics), the split in $\mathtt{start}$ (and join in $\mathtt{end}$) are correctly mined as an AND relation. Finally, the dependency graph in (c) shows the result obtained when enabling the discovery of duplicate tasks. As can be seen, Fodina provides extensive configuration options to drive the layout of the resulting process model, while all three models fit the given trace.

5.4.1.4 Step 6 (optional): Assure Task Reachability

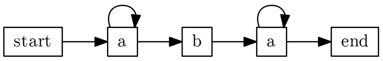
Next, step five (optional) assures that each task in the dependency graph is connected, as is a requirement for a valid Causal net. This is not implemented by checking if each task has at least one input and output arc in the dependency graph—similar as done by Heuristics Miner—but rather by continuing to add the next-best dependency graph until all tasks lie on a path between the start and end activity. Enabling this option almost completely reduces the effort involved when trying to find good threshold values, as well-scoring dependency arcs will be added until the dependency graph is valid. The implementation of assuring task connectedness in Fodina is more time consuming than the simple check performed by Heuristics Miner, but prevents the discovery of nets still containing disconnected elements, as was shown in Subsection 5.3.1.



(a) Dependency net obtained without mining for duplicate tasks and allowing “binary conflicts”. All splits and joins are in a XOR relation.



(b) Dependency net obtained without mining for duplicate tasks and with the “resolve binary conflict” option enabled, converting the length two loop to an AND split/join.



(c) Dependency net obtained with mining for duplicate tasks enabled.

Figure 5.7: Different dependency graph outcomes obtained with the Fodina miner under various configurations for the trace $\langle \mathtt{start}, a, a, a, b, a, a, a, \mathtt{end} \rangle$.

5.4.1.5 Step 7 (optional): Mine Long Distance Dependencies

The next step is also optional and mines long-distance dependencies from the event log, which is performed before the actual mining of the semantic AND and XOR relations in Fodina. We use the same dependency measure as the one described for Flexible Heuristics Miner [86], i.e.:

$$\left(\frac{2 \times (|a >>> b|)}{|a| + |b| + 1}\right) - \left(\frac{2 \times \mathit{abs}(|a| - |b|)}{|a| + |b| + 1}\right)$$

However, to better avoid the mining of unnecessary long-distance dependencies, we not only perform a check to see whether it is possible to go from a to the end task without visiting b (if b is always visited, the long-distance dependency is unnecessary), but also evaluate whether it is possible to go from the start to end task without visiting a (if a is always visited, the long-distance dependency is unnecessary) or without visiting b (similarly, if b is always visited, the long-distance dependency is unnecessary). Only if all these checks pass, the candidate long-distance dependency is introduced in the Causal net.

5.4.1.6 Step 8: Mine Split and Join Semantics

Finally, the dependency graph is converted to a Causal net by mining the AND and XOR relations to construct the input and output bindings. Our approach to do so is comparable to the pattern-based approach of Flexible Heuristics Miner, but adds configurable options to make the discovery more robust to noise.

To construct the output binding for a task, for example, we count the number of times each pattern (meaning a possible, particular subset of output tasks) was found after the occurrence of this task, but: (firstly) before the next occurrence of the task under consideration and (secondly) where the task under consideration was also the nearest input task for the pattern. If one of the output tasks is a long-distance dependent task, however, the nearest input check is not performed for this task, as the task under consideration can never be the nearest input (due to the dependency being long-distance). This method of including long-distance dependencies in the calculation of split and joins in not included in the description of the Flexible Heuristics Miner. Another improvement relates to the way patterns are selected for inclusion in the Causal net (recall that each pattern forms a different possible subset of activities in an AND relation in the output binding, with the subsets themselves being involved in a XOR relation). First, every pattern with a frequency ratio exceeding a configurable threshold is selected. Next, the remaining output tasks in the dependency graph which are *not* included in any output binding (in the thus-far selected patterns) are added as singleton subsets to the output binding. Configuring the pattern selection threshold thus also has an impact on the resulting Causal net's precision and generalization ability, as increasing the thresholds leads to the selection of less patterns (i.e. only the frequent patterns are selected), with potentially more output activities remaining which are then added as singleton subsets. Additionally, since the calculation of the splits and joins is the most computationally expensive step, we have implemented this step in a multi-threaded fashion, thus performing the calculation in a parallel manner.

Fodina has been implemented as a ProM 6 plugin. Apart from the aspects discussed above, the implemented algorithm also provides the ability to import and export Causal nets (one of the two Heuristics Miner implementations in ProM 6 currently lack this option), a visualizer which allows users to change the dependency thresholds mentioned above and immediately see the results in the dependency graph without having to restart the mining step, and a conversion plugin which allow to convert Causal nets to Petri nets. These technical additions will not be discussed in further detail in this work, as they do not provide theoretical background and can be inspected by running the implementation.

5.4.2 Causal Net Conformance Checking Metrics

5.4.2.1 Heuristic Execution Semantics

Next to the process discovery task, the process mining research field describes a second important analysis task, denoted as conformance checking, where existing process models are compared with behavior as captured in event logs so as to measure how well a process model performs with respect to the actual executions of the process at hand. As such, the "goodness" of a process model is typically assessed over the following quality dimensions: fitness (or: recall, sensitivity), indicating the ability of the process model to correctly execute the observed behavior; precision (or: appropriateness), i.e. the model's ability to disallow unwanted behavior; generalization, which indicates the model's ability to avoid overfitting and allow for unseen behavior; and finally, simplicity (or: structure, complexity), stating that simpler process models should be preferred above more complex ones if they are able to fit the observed behavior just as well.

In order to determine the quality of process models mined with Fodina in accordance with the given event log (or a new log), we first define an execution semantic for Causal nets, similar to the semantics used by the Improved Continuous Semantics (ICS) metric in Heuristics Miner [66]. Recall from the preliminary section that parsing a trace by a Causal net boils down to deriving a valid binding sequence, i.e. starting from the start task and ending with the end task without any pending obligations, and only removing pending obligations (consumed by the input binding chosen when executing a task). Also recall that the semantics of Causal nets are non-local, as an output binding may create the obligation to execute an activity much later in the process. Additionally, in the case of an output binding consisting of multiple sets of sets of tasks, it is not clear at the time of executing the task at hand which of the output bindings will be resolved later on. As such, to streamline and speed up the parsing of traces when "replaying" them in a Causal net, we implement *a heuristic replay procedure* as follows.

During the replay, a set of *pending obligations* is kept which must be fulfilled by future tasks. These obligations are nothing more than the set of output bindings derived from earlier executed tasks which still must be resolved: $PO = \{t_o | t \in$

$T_C, o \subseteq O(t)\}$. As such, PO is initially empty when replaying a trace σ. Next, all activities $\sigma_i \in \sigma$ are iterated. Each time an activity is fired, the best task candidate tasks among the duplicates in the Causal net is chosen (thus: $\mu(\sigma_i)$—recall the mapper function introduced above). Then, for the selected task which is to be fired, the best input binding $\in I(\mu(\sigma_i))$ is determined based on the input binding containing the least amount of unsatisfied ("missing") input tasks. An input task x for an input binding $\in I(\mu(\sigma_i))$ is unsatisfied when $\nexists t_o \in PO : [t = x \wedge \mu(\sigma_i) \in \bigcup(o)]$. Naturally, the most optimal input binding is one which has no missing input tasks and can thus fire without error; in the case where multiple input bindings can be satisfied, the largest one containing the most amount of tasks is selected. After firing, the set of pending obligations is updated. First of all, the fired task $\mu(\sigma_i)$ is removed from all pending obligations which still contain the fired task in one of their output bindings which still must be resolved. Formally:

$$\forall b \in o, t_o \in PO : [\mu(\sigma_i) \in \bigcup(o)] \text{ update binding:}$$
$$b = \begin{cases} b \setminus \mu(\sigma_i) & \text{if } \mu(\sigma_i) \in b \\ \emptyset & \text{if } \mu(\sigma_i) \notin b \end{cases}$$

Once a pending obligation $t_o \in PO$ is fulfilled (meaning that no activities are present in the pending output bindings: $\forall b \in o : [b = \emptyset]$), it is removed from the set of pending obligations altogether, i.e. $PO = PO \setminus t_o$. Finally, before moving on to the next activity, the list of pending obligations is updated with a new obligation containing the output bindings of the activity which was fired, i.e. $PO = PO \cup \{\mu(\sigma_i)_{O(\mu(\sigma_i))}\}$. At the end of trace replay, all leftover pending obligations are iterated and counted (using the pending output binding in each obligation having the least amount of tasks still pending) to calculate the number of remaining tokens, i.e. equal to $\sum_{t_o \in PO} |\mathrm{argmin}_{b \in o}(|b|)|$. Note that this replay procedure is heuristic, as it uses a local firing semantic to determine the best input set instead of considering each possible execution binding for a Causal net and see whether the trace can be found in such a binding (as this leads to an explosion of state for large Causal nets). Nevertheless, for Causal nets mined with Fodina (and other heuristic miners), this replay semantic is able to correctly parse the traces contained in the event log.

As a simple example, consider the dependency graph in Figure 5.7(b) once again. The trace $\langle \mathtt{start}, a, a, a, b, a, a, a, end \rangle$ is now replayed as follows:

TASK σ_i	$I(\sigma_i)$	$O(\sigma_i)$	PO
start	$\{\{\emptyset\}\}$	$\{\{a, b\}\}$	$\{start_{\{\{a,b\}\}}\}$
a	$\{\{\mathbf{start}\}, \{a\}\}$	$\{\{a\}, \{end\}\}$	$\{start_{\{\{b\}\}}, a_{\{\{a\},\{end\}\}}\}$
a	$\{\{start\}, \{\mathbf{a}\}\}$	$\{\{a\}, \{end\}\}$	$\{start_{\{\{b\}\}}, a_{\{\{a\},\{end\}\}}\}$
a	$\{\{start\}, \{\mathbf{a}\}\}$	$\{\{a\}, \{end\}\}$	$\{start_{\{\{b\}\}}, a_{\{\{a\},\{end\}\}}\}$
b	$\{\{\mathbf{start}\}\}$	$\{\{end\}\}$	$\{a_{\{\{a\},\{end\}\}}, b_{\{\{end\}\}}\}$
a	$\{\{start\}, \{\mathbf{a}\}\}$	$\{\{a\}, \{end\}\}$	$\{b_{\{\{end\}\}}, a_{\{\{a\},\{end\}\}}\}$
a	$\{\{start\}, \{\mathbf{a}\}\}$	$\{\{a\}, \{end\}\}$	$\{b_{\{\{end\}\}}, a_{\{\{a\},\{end\}\}}\}$
a	$\{\{start\}, \{\mathbf{a}\}\}$	$\{\{a\}, \{end\}\}$	$\{b_{\{\{end\}\}}, a_{\{\{a\},\{end\}\}}\}$
end	$\{\{\mathbf{a}, \mathbf{b}\}\}$	$\{\{\emptyset\}\}$	$\emptyset$

The first three columns depict the fired task and its input and output bindings. The boldfaced input binding denotes the choice made by the replay algorithm in order to fire the task. The final column contains the list of pending obligation. To fire the first start activity, for instance, the only available input binding is an empty set and hence, this activity can fire. The list of pending obligations is updated with the output bindings for start. Next, to fire a, both input bindings ($\{start\}$ and $\{a\}$) are evaluated to choose the most appropriate one. In this case, a is not present in the list of pending obligations, but start is and does still enable a (a is contained in one of the output bindings coupled to start). As such, this input binding is chosen. The list of pending obligations is updated as follows: first, a is removed from all the pending output bindings of start. Next, new obligations are added with the output bindings for a. To fire the second a task, both input bindings are evaluated again. Now, however, the pending output bindings for start are no longer able to satisfy the execution of a, since a is not contained anymore in any of the pending output bindings for start. However, the list of pending obligations now also contains $a_{\{\{a\},\{end\}\}}$, which does contain a in one of its output bindings. Hence, the $\{a\}$ input binding is chosen for the second a task. Once more, the list of pending obligations is updated as follows: the pending output bindings related to start do not contain a and hence remain untouched. The pending output bindings related to a ($\{a\}$ and $\{end\}$) do contain a and are thus updated in accordance with Equation (17): a is contained in $\{a\}$ and removed from this set, which then becomes empty; a is not contained in $\{end\}$ and as such this complete output binding emptied. All pending output bindings related to a are now empty, and hence, this obligation is removed from PO. Finally, a new set of obligations related to the just executed a task ($a_{\{\{a\},\{end\}\}}$) is added again. Considering the final $\mathtt{end}$ task, only one input binding ($\{a, b\}$) is available but is also enabled (both a and b are in the list of pending obligations and still allow $\mathtt{end}$ in one of their pending output bindings). The list of pending obligations is updated as follows: $b_{\{\{\mathtt{end}\}\}}$ still contains $\mathtt{end}$ in one of its pending output bindings and as such, $\mathtt{end}$ is removed from the (single) output binding, resulting in an empty set and discarding this obligation. $a_{\{\{a\},\{\mathtt{end}\}\}}$ also contains $\mathtt{end}$ in one of its pending output bindings and as such, $\mathtt{end}$ is removed from $\{\mathtt{end}\}$ and $\{a\}$ is emptied as this binding does not contain $\mathtt{end}$, hence also discarding this second pending obligation. This leaves an empty set of pending obligations. Note that all tasks in this trace can be fittingly executed.

5.4.2.2 Conformance Checking Metrics

Now that we have defined event-local execution semantics for Causal nets, various conformance checking metrics can be defined. First of all, we have reimplemented the Improved Continuous Semantics (ICS) metric to use with our defined execution semantics. The actual definition of the ICS metric itself is equal to the one applied by Heuristics Miner and its variants, i.e. equal to the fitness metric $PF_{complete}$ as described by Alves de Medeiros [66]. The ICS fitness after replaying a log L on a Causal net $C_N = (T_C, t_s, t_e, I, O)$ is thus:

$$ICS(L, C_N) = \frac{pTa - \left(\frac{mTo}{|L|-mTr+1} + \frac{rTo}{|L|-rTr+1}\right)}{\sum_{\sigma\in L}|\sigma|}$$

(range: $]-\infty, 1]$) with pTa ("parsed tasks") equal to the number of activities which could be successfully executed by the process model, mTo ("missing tokens")) equal to the total number of tokens which were missing while replaying the process log, rTo ("remaining tokens") equal to the total number of tokens which were remaining after replaying the process log, and with mTr ("missing traces") and rTr ("remaining traces") equal to the number of traces where tokens were missing, or remaining after the trace replay respectively. Note that—using our defined execution semantics—a missing token is created for each task in the chosen input binding of a fired task which was not present in the list of pending obligations. Similarly, a remaining token is created for each task belonging to the smallest output binding coupled to a pending obligation. For instance, if the final set of pending obligations equals $\{a_{\{\{x\}\}}, b_{\{\{x\},\{y,z\}\}}\}$, two missing tokens are created (one for $\{x\}$ as the smallest pending output binding for a and one for $\{x\}$ as the smallest pending output binding for b).

Next, we also define two more coarse-grained metrics which operate on the dependency graph (T_C, D) defined over $C_N = (T_C, t_s, t_e, I, O)$ with $D = \{(a,b) | a \in T_C \wedge b \in T_C \wedge (a \in \Box b \vee b \in a\Box)\}$. The first metric just defines a straightforward flow evaluation based on the presence of arcs in the dependency graph:

$$Flow(L, (T_C, D)) = \frac{pFlows}{pFlows + upFlows}$$

(range: $[0,1]$) with $pFlows$ ("parsed flows") $= |\{(\sigma_i, \sigma_{i+1}) | \sigma \in L, 1 \leqslant i < |\sigma|, (\mu(\sigma_i), \mu(\sigma_{i+1})) \in D\}|$ and $upFlows$ ("unparsed flows") $= |\{(\sigma_i, \sigma_{i+1}) | \sigma \in L, 1 \leqslant i < |\sigma|, (\mu(\sigma_i), \mu(\sigma_{i+1})) \notin D\}|$.

The second metric defines a fuzzy fitness evaluation, by parsing the dependency graph in a similar way as the semantics defined by Fuzzy Miner [67, 157]:

$$Fuzzy(L, (T_C, D)) = \frac{pTasks}{pTasks + upTasks}$$

(range: $[0,1]$) with $pTasks$ ("parsed tasks") the number of tasks which could be parsed by the dependency graph and $upTasks$ ("unparsed tasks") the tasks which could not. To determine which activities in a trace σ can be parsed, a set of enabled tasks is constructed, initially set to an empty set. Next, for each activity $\sigma_i \in \sigma$, it is checked whether $\mu(\sigma_i)$ is present in the set of enabled tasks, or whether the corresponding task in the dependency graph contains no incoming arcs. If so, this activity σ_i is parseable and removed from the set of enabled tasks. Next, the tasks $\{b | b \in T_C | (\sigma_i, b) \in D\}$ are added to the set of enabled tasks and the following activity in the trace is evaluated. This corresponds to the "AND split; memoryless XOR join" semantics as defined in Fuzzy Miner.

As we have defined event-local execution semantics, the possibility exists to re-utilize existing conformance checking metrics which depend only on such semantics (i.e. determining whether an activity in a trace can be parsed by the model or not)—even although their original implementation may assume another representational language to represent process models. We refer for example to our conformance checking metrics presented in Chapter 3. These metrics can directly be applied to our proposed approach, since our defined execution semantics allow to determine for each $a \in A_L$, given a list of pending obligations, whether this activity can be executed fittingly or not.

This allows us to directly define an event-granular fitness metric for causal nets as follows:

$$\mathrm{Recall}(L, C_N) = \frac{pTasks}{pTasks + upTasks}$$

(range: $[0, 1]$) with $pTasks$ ("parsed tasks") again the number of parsed tasks and $upTasks$ ("unparsed tasks") the tasks which could not, but now based on the replay procedure executed on the Causal net. Recall that a task is unparseable whenever no satisfied input binding can be selected for the task to be executed.

Finally, we also remark that the discovered Causal nets can be converted to Petri nets, which allows for a plethora of other conformance checking metrics available in literature to be applied.

The following section describes a thorough evaluation experiment to benchmark Fodina against related heuristic process discovery algorithms. For this experiment, we include two additional metrics which can be constructed over general, event-granular conformance checking techniques. The first is the well known "parsing measure", which returns the ratio of traces which could be parsed without errors according to the execution semantics of a certain process model, further on denoted as PM (range: $[0, 1]$). We also define a "Fitting Single Trace Measure", which also returns the percentage of traces which could be correctly parsed according to the execution semantics of a certain process model, with the difference that, here, a model is mined for each single trace in the event log which is subsequently used in the conformance evaluation. This metric is further denoted as $PM^1 = \frac{|\{PM(\sigma)=1|\sigma \in L\}|}{|L|}$ (range: $[0, 1]$) and will be used as a basic robustness check, to determine whether the evaluated discovery algorithms succeed in mining fitting process models for event logs containing a single trace, as discussed in Subsection 5.3.2.

5.5 Experimental Evaluation

This section presents the results of a thorough evaluation experiment which was performed in order to compare the performance, robustness and scalability of Fodina with related heuristic process discovery algorithms.

5.5.1 Experimental Setup

5.5.1.1 Input Logs

We have included 50 different event logs to perform the experimental evaluation (see Table 5.2). Logs "a10skip" to "l2lskip" are commonly used artificial event logs [66] (type: "synthetic"). Logs "prAm6" to "prGm6" are also synthetic and have been utilized in a recent benchmarking study by Munoz-Gama et al. [158]. Next, logs "perml$X$aY" (type: "permutations") contain all permutations (with repetition) of length X with number of activity types equal to Y (the log size is hence Y^X). The best model for these logs is obviously a "flower model" which allows any sequence of activities, but these logs will be used to perform robustness checks by iterating over and mining each trace separately. Logs "randpms$X$dY" are logs with size X generated from a randomly constructed process model[8] with depth Y (type: "random"). Logs "rands$A$lBmCaD" are also randomly generated, but purely by choosing random activities out of a set with size D to construct A traces with mean length B and standard deviation C, i.e. not simulated from a (random) process model. Logs "real$X$" finally encompass four real-life logs (type: "real").

Table 5.2 also provides an overview of the structural characteristics for the event logs included in the experiment, depicting the number of traces $|L|$, number of distinct traces $|\bigcup(L)|$, number of activities in the log $|T_L|$ together with the amount of activities acting as starting or ending activities in a trace (most event logs already include unique starting and ending activities) and, finally, the minimum, average and maximum trace length, together with the standard deviation.

5.5.1.2 Discovery Techniques

The following discovery algorithms are considered in the experimental setup:

- α-algorithm [55, 58, 68, 160]: using the "Alpha Miner" (abbreviated as A in the following tables) and "Alpha Miner++" (A++) plugins in ProM 5.2, and using the "Alpha Miner" plugin in ProM 6 ($\mathsf{A6}$). The native output process model representation for this algorithm is a Petri net.

[8]Using the Process Log Generator [159], see: `http://www.processmining.it/sw/plg`.

Table 5.2: Structural log characteristics for event logs included in experimental setup. Remark that some tabulated means or trace lengths are higher then the configured values for the "permutations" and "random" logs, as artificial start and end tasks were added before and after each generated trace, thus increasing the length of each trace by 2.

Event Log	Type	$\lvert L \rvert$	$\lvert \bigcup(L) \rvert$	$\lvert A_L \rvert$	Min. Trace Length	Avg. Trace Length	Max. Trace Length	Trace Length StdDev.
a10skip	synthetic	300	6	12	8.00	8.88	10.00	0.72
a12	synthetic	300	5	14	7.00	8.31	9.00	0.81
a5	synthetic	300	13	7	6.00	6.78	12.00	1.19
a6nfc	synthetic	300	3	8	6.00	6.80	7.00	0.40
a7	synthetic	300	14	9	6.00	6.74	7.00	0.44
a8	synthetic	300	4	10	5.00	6.01	8.00	1.42
betasimplified	synthetic	300	4	13	11.00	12.03	13.00	0.71
choice	synthetic	300	16	12	8.00	8.00	8.00	0.00
driverslicense	synthetic	2	2	9	7.00	7.00	7.00	0.00
driverslicenseloop	synthetic	350	87	11	9.00	13.53	21.00	4.82
herbstfig3p4	synthetic	32	32	12	9.00	17.34	45.00	8.23
herbstfig5p19	synthetic	300	6	8	4.00	6.09	8.00	2.00
herbstfig6p18	synthetic	300	153	7	6.00	16.26	53.00	9.96
herbstfig6p31	synthetic	300	4	9	6.00	6.00	6.00	0.00
herbstfig6p36	synthetic	300	2	12	10.00	10.00	10.00	0.00
herbstfig6p38	synthetic	300	5	7	8.00	8.00	8.00	0.00
herbstfig6p41	synthetic	300	12	16	12.00	12.00	12.00	0.00
l2l	synthetic	300	10	6	5.00	7.14	27.00	3.07
l2loptional	synthetic	300	9	6	4.00	6.09	20.00	2.83
l2lskip	synthetic	300	8	6	6.00	7.84	24.00	2.64
prAm6	synthetic	1200	1049	363	19.00	31.63	41.00	4.16
prBm6	synthetic	1200	1126	317	14.00	41.49	59.00	11.04
prCm6	synthetic	500	500	311	15.00	42.93	59.00	11.35
prDm6	synthetic	1200	1200	429	235.00	248.61	271.00	10.38
prEm6	synthetic	1200	1200	275	80.00	98.76	116.00	8.12
prFm6	synthetic	1200	1200	299	234.00	240.78	245.00	2.02
prGm6	synthetic	1200	1200	335	124.00	143.07	159.00	6.54
perml10a3	permutations	59049	59049	5	12.00	12.00	12.00	0.00
perml3a10	permutations	1000	1000	12	5.00	5.00	5.00	0.00
perml3a3	permutations	27	27	5	5.00	5.00	5.00	0.00
perml3a5	permutations	125	125	7	5.00	5.00	5.00	0.00
perml5a10	permutations	100000	100000	12	7.00	7.00	7.00	0.00
perml5a3	permutations	243	243	5	7.00	7.00	7.00	0.00
perml5a5	permutations	3125	3125	7	7.00	7.00	7.00	0.00
randpms10000d1	random	10000	2	8	8.00	8.00	8.00	0.00
randpms10000d2	random	10000	4724	16	10.00	33.45	275.00	27.73
randpms10000d3	random	10000	2906	38	10.00	21.96	141.00	15.06
randpms1000d1	random	1000	17	7	6.00	12.44	57.00	7.89
randpms1000d2	random	1000	12	14	14.00	14.00	14.00	0.00
randpms1000d3	random	1000	998	51	35.00	55.75	238.00	24.88
randpms100d1	random	100	3	9	7.00	7.00	7.00	0.00
randpms100d2	random	100	55	18	8.00	23.54	117.00	20.49
randpms100d3	random	100	54	20	18.00	21.21	45.00	4.42
rands10000l20m8a10	random	10000	10000	12	14.00	21.53	29.00	4.62
rands1000l10m4a5	random	1000	999	7	8.00	11.57	15.00	2.28
rands100l5m2a3	random	100	90	5	5.00	6.65	8.00	1.11
realdocman	reallife	12391	1411	70	5.00	5.30	11.00	0.60
realhospital	reallife	1143	981	626	3.00	133.49	1816.00	202.62
realincman	reallife	24770	1174	18	3.00	5.01	29.00	1.94
realoutsourcing	reallife	276599	3151	7	2.00	4.14	40.00	1.59

- Heuristics Miner [44, 57]: using the “Heuristics Miner” plugin in ProM 5.2 (HM5) and the “Mine for a Heuristics Net using Heuristics Miner” plugin in ProM 6 (HM6); HM5_L and HM6_L describe the configuration where the dependency thresholds have been set to their lowest values. The native output process model representation for this algorithm is a Heuristics net.
- Flexible Heuristics Miner [86]: using the “Mine for a Causal Net using Heuristics Miner” plugin in ProM 6 (FHM6); FHM6_L describes the configuration where the dependency thresholds have been set to their lowest values. The native output process model representation for this algorithm is a Causal net (internally represented in ProM as a “Flex net”).
- Fodina: using our implemented ProM 6 plugin (F), also with a low dependency threshold variant, F_L. The other parameters were kept default (no duplicate task mining and no binary conflict resolution). The native output process model representation for this algorithm is a Causal net.

Note that we do not include the “Tsinghua Alpha Miner” plugin provided by ProM 5, as this algorithm derives disconnected, faulty nets when no distinct start and complete event types are available. We also do not include Heuristics Miner++ [85] and the Stream-aware Heuristics Miner [87] in our experimental setup, as they operate mostly similar as Heuristics Miner in ProM 5.2 or only perform additional log pre-processing steps without modifying the discovery procedure. Finally, we do not include Fuzzy Miner [67, 157], due to the following reasons. First, although the implemented Fuzzy Miner plugin displays a “log conformance” percentage, indicating which percentage of traces could be replayed correctly by the shown Fuzzy model, it is hard to compare the result given by this metric with other conformance checking metrics, and hence discovery algorithms. Second, in Fuzzy Miner, users have the option to configure various thresholds in order to show less or more edges or cluster activities together, impacting the shown conformance metric. Third, choosing to show a model including all behavior results in a Fuzzy model with a perfect conformance score. Although this model is overfitting, Fuzzy Miner itself does not report a precision related conformance metric. As it is not possible to convert a Fuzzy model to another process representation (such as Petri nets), precision and generalization assessments cannot be derived.

5.5.1.3 Model Conversion

Apart from working with the native output of each discovery algorithm, we also apply various conversion plugins to convert a process model to other representations. Figure 5.8 provides an overview of the conversion possibilities currently offered and implemented in ProM. This leads to the following possibilities regarding process model formats:

- Petri net (use as discovered by α-algorithms).
- Heuristics net (use as discovered by Heuristics Miner).
- Flex net (use as discovered by Flexible Heuristics Miner).
- Causal net (use as discovered by Fodina).
- Causal net $\rightarrow$ Petri net (using our conversion plugin).
- Causal net $\rightarrow$ Flex net (using our conversion plugin).
- Causal net $\rightarrow$ Flex net $\rightarrow$ Petri net (using our conversion plugin followed by a plugin by Adriansyah [109]).
- Flex net $\rightarrow$ Petri net (using a plugin by Adriansyah [109]).
- Flex net $\rightarrow$ Causal net (using our conversion plugin).
- Flex net $\rightarrow$ Causal net $\rightarrow$ Petri net (using our conversion plugins).
- Heuristics net $\rightarrow$ Petri net (using our conversion plugin, this ensures a correct Petri net).
- Heuristics net $\rightarrow$ Petri net (using built in convertor [86], this results in an incorrect Petri net for Heuristic nets mined with ProM 6).
- Heuristics net $\rightarrow$ Flex net (using a plugin by Ribeiro [86]).
- Heuristics net $\rightarrow$ Flex net $\rightarrow$ Petri net (using a plugin by Ribeiro [86] followed by a plugin by Adriansyah [109]).
- Heuristics net $\rightarrow$ Flex net $\rightarrow$ Causal net (using a plugin by Ribeiro [86] followed by our conversion plugin).
- Heuristics net $\rightarrow$ Flex net $\rightarrow$ Causal net $\rightarrow$ Petri net (using a plugin by Ribeiro [86] followed by our conversion plugins).

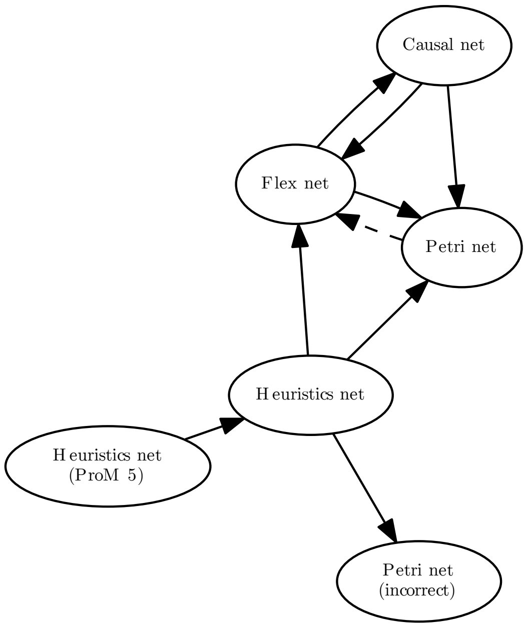


Figure 5.8: Conversion possibilities between process model representation formats.

Naturally, when required, we start each conversion from a “native” output and do not allow conversion “loops”. Remark that a plugin is available in ProM 6 to convert Petri nets to a Flexible model, but this conversion induces routing tasks which make the resulting model unusable for Causal net related metrics. We have not implemented a separate Petri net to Causal net convertor.

5.5.1.4 Conformance Checking Metrics

We use the following conformance checking metrics in the experimental setup to evaluate the discovered process models by the included discovery algorithms:

- For Causal nets: the ICS Fitness (ICS).

- For Flex nets: the Average Alignment Based Trace Fitness (f_a^{avg}) [108–111], using the alignment-based replaying plugin directly on the Flex net. Note however that internally, the plugin converts the Flex net to a Petri net to perform the actual alignment.
- For Heuristic nets: the ICS Fitness [66] (ICS).
- For Petri nets: the Average Alignment Based Trace Fitness (f_a^{avg}), One Align Precision (a_p^1), Best Align Precision (a_p), ETC Precision (etc_P), Behavioral Recall (r_B), Weighted Behavioral Precision (p_B^w), Weighted Behavioral Generalization (g_B^w), Number of Arcs, Places and Transitions ($\#a$, $\#p$, $\#t$).
- For all native models: Fitting Single Trace Measure (PM^1) using the discovery algorithm's native model's execution semantics.

Since exploring all possible model conversion paths as mentioned above leads to an explosions of model-log pairs to test, we limit ourselves to the following model formats:

- Causal nets: native and converted to Petri net using our conversion plugin.
- Flex nets: native, converted to Petri net using a plugin by Adriansyah [109] and converted to Causal nets using our conversion plugin.
- Heuristic nets: native and converted to Petri net using a *correct* conversion method (using our plugin where necessary; note thus that the benchmark results will differ from the results obtain by standard ProM 6 use).
- Petri nets: native only.

5.5.2 Results

5.5.2.1 Benchmark Comparison

Table 5.3 lists the main results of our benchmarking experiment for the ICS Fitness, Behavioral Recall and Weighted Behavioral Precision metrics, whereas Table 5.4 list results for the simplicity metrics. Full experimental result tables are appended at the end of this chapter. As can be seen from the results, Fodina is able to outperform other heuristic, dependency-based miners. A number of important remarks should be kept in mind when interpreting the results. First, a metric score below the maximum does not indicate Fodina's inability to obtain a fitting process model, as most configuration parameters have been kept default (apart from dependency thresholds). Second, a value of NA indicates a timeout during the conformance checking procedure, and does not necessarily signpost a discovery algorithm crash. Third, keep in mind the importance of fitness and simplicity metrics above precision and generalization, as the latter two can display a maximum score for process models which could not be mined by the discovery algorithm (for example, a process model which is unable to replay any trace will display a high precision score in most cases).

5.5.2.2 Robustness

Table 5.5 lists the results of Fodina for the Fitting Single Trace Measure metric. Recall that this metric corresponds to the percentage of traces in the event log for which a fitting model could be mined, for this trace in isolation.

This metric is included as a simple robustness check, but, surprisingly, we observe from the results that only Fodina is able to mine all single traces correctly. Note that we relied on the most relaxed configuration parameters regarding dependency thresholds for each miner.

5.5.2.3 Scalability

The graphs in Figure 5.9 depict the results of a scalability experiment which was performed to compare the performance in terms of speed of Fodina with Flexible Heuristics Miner and Heuristics Miner (all with default settings). For both algorithms, the ProM 6 implementation was chosen. The run times of the compared algorithms were assessed over the following dimensions (each graph allows to inspect a pair of dimensions). First, the total log size $|L|$, ranging from 10 to 10000 traces. Second, the distinct log size $|\bigcup L|$ (always smaller than the total log size), ranging from 5 to 1000 distinct traces. Third, the number of activities occurring in the log, $|A_L|$, ranging from 2 to 25, and finally, the length of the traces contained in the log, $|\sigma|$ with $\sigma \in L$, ranging from 3 to 50. All runs were repeated twenty times. The results show that Fodina is able to outperform other heuristic-based miners. Even although Fodina is theoretically bounded by

Table 5.3: Results of the Fodina conformance checking experiment. This table lists the results for the ICS Fitness metric.

Algorithm	HM5	HM6	FHM6	F	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	hnet	hnet	cnet	cnet	hnet	hnet	cnet	cnet
Event Log				ICS				
permutations\perml10a3.xes	0.996	0.996	0.959	0.996	0.996	0.894	1.000	1.000
permutations\perml3a10.xes	-2.116	-2.116	0.969	0.998	-2.116	-2.116	1.000	1.000
permutations\perml3a3.xes	-0.200	0.565	0.944	0.527	-0.200	0.772	1.000	1.000
permutations\perml3a5.xes	-0.224	-0.224	0.952	0.990	-0.224	-0.224	1.000	1.000
permutations\perml5a10.xes	-0.830	-0.830	0.960	0.929	-0.830	-0.830	1.000	1.000
permutations\perml5a3.xes	0.945	0.945	0.943	0.945	0.945	0.881	1.000	1.000
permutations\perml5a5.xes	0.859	0.859	0.944	0.986	0.859	0.854	1.000	1.000
random\randpms10000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.875	1.000
random\randpms10000d2.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms10000d3.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	1.000	1.000	1.000	1.000	1.000	0.741	0.929	1.000
random\randpms1000d3.xes	1.000	1.000	1.000	1.000	1.000	0.960	0.845	1.000
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.998	1.000
random\randpms100d3.xes	1.000	1.000	1.000	1.000	1.000	0.979	0.929	1.000
random\rands10000l20m8a10.xes	0.964	0.964	0.999	0.992	0.964	-2.497	0.570	1.000
random\rands1000l10m4a5.xes	0.952	0.952	0.994	0.989	0.952	-0.392	0.744	1.000
random\rands100l5m2a3.xes	0.913	0.913	0.958	0.970	0.913	-0.233	0.994	1.000
reallife\realdocman.xes	0.662	0.669	0.986	0.967	0.662	0.399	0.998	0.996
reallife\realhospital.xes	0.768	0.606	0.979	0.969	0.768	0.480	0.999	0.999
reallife\realincman.xes	0.823	0.795	0.959	0.965	0.823	0.552	1.000	0.998
reallife\realoutsourcing.xes	NA	-2.844	0.500	0.999	NA	-3.781	0.513	0.999
synthetic\a10skip.xes	1.000	1.000	1.000	1.000	1.000	0.945	1.000	1.000
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a5.xes	1.000	1.000	1.000	1.000	1.000	0.937	0.999	1.000
synthetic\a6nfc.xes	1.000	1.000	1.000	0.990	1.000	1.000	0.998	0.990
synthetic\a7.xes	0.957	0.999	0.999	0.950	0.957	NA	0.998	0.994
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\betasimplified.xes	1.000	1.000	1.000	1.000	1.000	0.961	1.000	1.000
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	0.923	1.000	1.000
synthetic\driverslicense.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicenseloop.xes	1.000	1.000	1.000	1.000	1.000	0.866	1.000	1.000
synthetic\herbstfig3p4.xes	1.000	1.000	1.000	0.998	1.000	1.000	0.942	1.000
synthetic\herbstfig5p19.xes	1.000	1.000	1.000	1.000	1.000	0.671	0.999	1.000
synthetic\herbstfig6p18.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p31.xes	0.500	1.000	1.000	1.000	0.500	1.000	1.000	1.000
synthetic\herbstfig6p36.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p38.xes	0.552	0.552	0.875	0.963	0.552	-0.066	0.999	1.000
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.999	1.000
synthetic\l2l.xes	0.848	1.000	1.000	1.000	0.848	1.000	1.000	1.000
synthetic\l2loptional.xes	0.911	1.000	1.000	1.000	0.911	0.917	1.000	1.000
synthetic\l2lskip.xes	0.882	1.000	1.000	1.000	0.882	1.000	1.000	1.000
synthetic\prAm6.xes	0.924	0.936	0.972	0.952	0.924	NA	0.990	0.999
synthetic\prBm6.xes	0.966	0.979	0.991	0.970	0.966	0.979	0.992	0.998
synthetic\prCm6.xes	-0.245	-0.291	0.053	0.058	-0.245	-0.338	0.669	0.981
synthetic\prDm6.xes	-0.187	-0.424	0.060	-0.124	-0.187	-0.487	0.527	1.000
synthetic\prEm6.xes	0.220	0.073	0.464	0.394	0.220	0.006	0.569	0.999
synthetic\prFm6.xes	0.324	0.326	0.509	0.467	0.324	0.315	0.450	1.000
synthetic\prGm6.xes	0.086	-0.155	0.371	0.308	0.086	-0.241	0.571	0.999

Table 5.3 (continued): Results of the Fodina conformance checking experiment. This table lists the results for the Behavioral Recall metric.

Algorithm	A	A++	A6	HM5	HM6	FHM6	F	$HM5_L$	$HM6_L$	$FHM6_L$	F_L
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						r_B					
permutations\perml10a3.xes	1.000	1.000	NA	0.996	0.996	0.699	0.996	0.996	0.917	1.000	1.000
permutations\perml3a10.xes	1.000	1.000	1.000	0.800	0.800	0.721	0.732	0.800	0.800	1.000	1.000
permutations\perml3a3.xes	1.000	1.000	1.000	0.600	0.667	0.815	0.689	0.600	0.844	1.000	1.000
permutations\perml3a5.xes	1.000	1.000	1.000	0.800	0.800	0.854	0.848	0.800	0.800	1.000	1.000
permutations\perml5a10.xes	1.000	1.000	1.000	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	1.000	1.000	1.000	1.000	1.000	1.000	0.999	1.000
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	0.714	0.929	0.714	1.000	1.000	1.000	1.000	1.000	0.914	1.000	1.000
random\randpms1000d3.xes	0.725	0.771	0.725	1.000	1.000	1.000	1.000	1.000	0.962	0.996	1.000
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.841	0.884	0.841	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d3.xes	0.808	0.906	0.808	1.000	1.000	1.000	1.000	1.000	0.979	0.994	1.000
random\rands10000l20m8a10.xes	1.000	1.000	1.000	0.954	0.954	NA	0.911	0.954	NA	NA	1.000
random\rands1000l10m4a5.xes	1.000	1.000	1.000	0.914	0.914	0.863	0.910	0.914	0.764	0.849	1.000
random\rands100l5m2a3.xes	1.000	1.000	1.000	0.917	0.917	0.881	0.868	0.917	0.669	0.916	1.000
reallife\realdocman.xes	NA	NA	NA	0.666	0.673	NA	0.967	0.668	0.626	NA	0.991
reallife\realhospital.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
reallife\realincman.xes	0.630	0.349	0.631	0.818	0.794	0.885	0.965	0.809	0.584	NA	0.998
reallife\realoutsourcing.xes	0.348	0.002	0.004	NA	0.008	0.827	0.977	NA	0.126	0.695	0.977
synthetic\a10skip.xes	0.946	0.946	0.946	1.000	1.000	1.000	1.000	1.000	0.946	1.000	1.000
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a5.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.937	1.000	1.000
synthetic\a6nfc.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.990	1.000	1.000	0.990	0.990
synthetic\a7.xes	1.000	1.000	1.000	0.957	1.000	0.981	0.940	0.957	NA	0.985	0.994
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\betasimplified.xes	0.836	0.881	0.836	1.000	1.000	1.000	1.000	1.000	0.961	1.000	1.000
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.924	1.000	1.000
synthetic\driverslicense.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicenseloop.xes	NA	0.991	0.917	1.000	1.000	1.000	1.000	1.000	0.869	1.000	1.000
synthetic\herbstfig3p4.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig5p19.xes	0.743	0.828	0.743	1.000	1.000	1.000	1.000	1.000	0.836	1.000	1.000
synthetic\herbstfig6p18.xes	NA	0.795	0.734	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p31.xes	1.000	1.000	1.000	0.833	1.000	1.000	1.000	0.833	1.000	1.000	1.000
synthetic\herbstfig6p36.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p38.xes	1.000	1.000	1.000	0.875	0.875	1.000	0.963	0.875	0.750	1.000	1.000
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.984	1.000
synthetic\l2l.xes	1.000	1.000	1.000	0.850	1.000	1.000	1.000	0.850	1.000	1.000	1.000
synthetic\l2loptional.xes	1.000	1.000	1.000	0.911	1.000	1.000	1.000	0.911	0.918	1.000	1.000
synthetic\l2lskip.xes	0.872	1.000	0.872	0.883	1.000	1.000	1.000	0.883	1.000	1.000	1.000
synthetic\prAm6.xes	0.869	NA	0.869	0.923	0.936	0.939	0.941	0.923	NA	NA	0.999
synthetic\prBm6.xes	0.982	0.983	0.982	0.968	0.979	0.968	0.956	0.968	0.979	0.954	0.998
synthetic\prCm6.xes	NA	NA	NA	0.614	0.606	0.594	0.685	0.614	0.602	NA	NA
synthetic\prDm6.xes	0.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prEm6.xes	NA	NA	0.373	0.768	0.726	NA	0.765	0.766	0.703	NA	NA
synthetic\prFm6.xes	0.000	NA	NA	NA	0.784	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	0.000	NA	NA	0.733	NA	NA	NA	0.733	NA	NA	NA

Table 5.3 (continued): Results of the Fodina conformance checking experiment. This table lists the results for the Weighted Behavioral Precision metric.

Algorithm	A	A++	A6	HM5	HM6	FHM6	F	$HM5_L$	$HM6_L$	$FHM6_L$	F_L
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						p_B^w					
permutations\perml10a3.xes	0.609	0.764	NA	0.893	0.893	0.804	0.893	0.893	0.933	0.815	0.815
permutations\perml3a10.xes	0.211	0.395	0.211	0.407	0.407	0.526	0.577	0.407	0.407	0.429	0.429
permutations\perml3a3.xes	0.393	0.573	0.393	0.783	0.852	0.680	0.861	0.783	0.691	0.648	0.648
permutations\perml3a5.xes	0.313	0.500	0.313	0.559	0.559	0.625	0.619	0.559	0.559	0.556	0.556
permutations\perml5a10.xes	0.290	0.531	0.290	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	0.929	0.929	NA	0.929	0.929	0.929	NA	0.791
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	1.000	0.764	1.000	0.830	0.830	0.830	0.830	0.830	0.608	0.613	0.753
random\randpms1000d3.xes	0.789	0.732	0.789	0.709	0.709	NA	0.709	0.709	0.350	NA	0.542
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.867	0.838	0.867	0.886	0.886	0.886	0.886	0.886	0.886	0.620	0.827
random\randpms100d3.xes	0.984	0.699	0.984	0.896	0.896	0.896	0.896	0.896	0.775	0.733	0.753
random\rands10000l20m8a10.xes	0.149	0.161	0.149	0.170	0.170	NA	NA	0.170	NA	NA	NA
random\rands1000l10m4a5.xes	0.324	0.389	0.324	0.437	0.435	0.411	0.430	0.435	0.413	0.413	0.403
random\rands100l5m2a3.xes	0.439	0.597	0.439	0.712	0.712	0.674	0.696	0.712	0.681	0.663	0.656
reallife\realdocman.xes	NA	NA	NA	NA	NA	NA	0.391	NA	NA	NA	NA
reallife\realhospital.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
reallife\realincman.xes	0.232	0.559	0.161	0.382	0.399	NA	0.681	0.375	0.324	NA	0.479
reallife\realoutsourcing.xes	0.896	0.089	0.086	NA	NA	NA	0.921	NA	NA	NA	NA
synthetic\a10skip.xes	0.946	0.946	0.946	1.000	1.000	1.000	1.000	1.000	0.946	0.789	0.900
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.803	0.853
synthetic\a5.xes	1.000	1.000	0.575	1.000	1.000	1.000	1.000	1.000	0.972	0.797	0.819
synthetic\a6nfc.xes	0.870	1.000	0.870	0.870	0.870	0.870	0.808	0.870	0.870	0.873	0.808
synthetic\a7.xes	1.000	1.000	1.000	0.967	0.804	0.826	0.944	0.967	NA	0.712	0.699
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.846	0.920
synthetic\betasimplified.xes	0.683	0.601	0.683	0.850	0.850	0.850	0.850	0.850	0.805	0.850	0.850
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.801	1.000	1.000
synthetic\driverslicense.xes	0.903	1.000	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903
synthetic\driverslicenseloop.xes	NA	0.921	0.899	0.893	0.893	0.893	0.893	0.893	0.647	0.593	0.691
synthetic\herbstfig3p4.xes	0.992	0.992	0.992	0.992	0.992	0.992	0.983	0.992	0.992	0.842	0.820
synthetic\herbstfig5p19.xes	0.873	0.885	0.873	0.902	0.902	0.902	0.902	0.902	0.886	0.802	0.776
synthetic\herbstfig6p18.xes	NA	0.773	0.466	0.971	0.971	0.971	0.971	0.971	0.971	0.971	0.971
synthetic\herbstfig6p31.xes	0.558	1.000	0.558	0.513	0.558	0.558	0.558	0.513	0.558	0.558	0.558
synthetic\herbstfig6p36.xes	0.976	1.000	0.976	0.976	0.976	0.976	0.976	0.976	0.976	0.976	0.976
synthetic\herbstfig6p38.xes	0.428	0.510	0.428	0.836	0.836	0.626	0.578	0.836	NA	0.719	0.598
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.613	0.741
synthetic\l2l.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	NA	1.000	1.000	1.000
synthetic\l2loptional.xes	1.000	1.000	1.000	0.873	1.000	1.000	1.000	0.873	0.918	1.000	1.000
synthetic\l2lskip.xes	1.000	0.887	1.000	0.829	1.000	1.000	1.000	0.829	1.000	1.000	1.000
synthetic\prAm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prBm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prCm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prDm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prEm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prFm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Table 5.4: Results of the Fodina conformance checking experiment for the simplicity metrics (number of arcs, places and transitions respectively).

Algorithm	A	A++	A6	HM5	HM6	FHM6	F	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml
Event Log						#a / #p / #t					
permutations\perml10a3.xes	2 / 2 / 5	10 / 3 / 5	NA / NA / NA	26 / 11 / 11	26 / 11 / 11	72 / 19 / 31	26 / 11 / 11	26 / 11 / 11	62 / 19 / 17	70 / 25 / 35	40 / 10 / 20
permutations\perml3a10.xes	2 / 2 / 12	24 / 3 / 12	2 / 2 / 12	82 / 32 / 32	82 / 32 / 32	1374 / 54 / 402	1294 / 34 / 362	82 / 32 / 32	82 / 32 / 32	504 / 144 / 252	264 / 24 / 132
permutations\perml3a3.xes	2 / 2 / 5	10 / 3 / 5	2 / 2 / 5	10 / 4 / 5	14 / 8 / 5	72 / 19 / 31	36 / 10 / 13	10 / 4 / 5	39 / 14 / 13	70 / 25 / 35	40 / 10 / 20
permutations\perml3a5.xes	2 / 2 / 7	14 / 3 / 7	2 / 2 / 7	42 / 17 / 17	42 / 17 / 17	214 / 29 / 77	174 / 19 / 57	42 / 17 / 17	42 / 17 / 17	154 / 49 / 77	84 / 14 / 42
permutations\perml5a10.xes	2 / 2 / 12	24 / 3 / 12	2 / 2 / 12	NA / NA / 32	NA / NA / 32	NA / NA / 1326	NA / NA / 452	NA / NA / 32	NA / NA / 32	NA / NA / 252	NA / NA / 132
permutations\perml5a3.xes	NA / NA / 5	NA / NA / 5	NA / NA / 5	NA / NA / 11	NA / NA / 11	NA / NA / 31	NA / NA / 11	NA / NA / 11	NA / NA / 17	NA / NA / 35	NA / NA / 20
permutations\perml5a5.xes	NA / NA / 7	NA / NA / 7	NA / NA / 7	NA / NA / 17	NA / NA / 17	NA / NA / 89	NA / NA / 49	NA / NA / 17	NA / NA / 33	NA / NA / 77	NA / NA / 42
random\randpms10000d1.xes	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 22	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 26	NA / NA / 14
random\randpms10000d2.xes	NA / NA / 0	NA / NA / 16	NA / NA / 16	NA / NA / 18	NA / NA / 18	NA / NA / 56	NA / NA / 18	NA / NA / 18	NA / NA / 18	NA / NA / 56	NA / NA / 18
random\randpms10000d3.xes	NA / NA / 38	NA / NA / 38	NA / NA / 38	96 / 40 / 43	96 / 40 / 43	270 / 127 / 130	96 / 40 / 43	96 / 40 / 43	96 / 40 / 43	346 / 134 / 162	146 / 46 / 73
random\randpms1000d1.xes	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	42 / 21 / 21	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	42 / 21 / 21	14 / 7 / 7
random\randpms1000d2.xes	74 / 25 / 14	33 / 14 / 14	78 / 26 / 14	54 / 20 / 26	54 / 20 / 26	114 / 50 / 56	54 / 20 / 26	54 / 20 / 26	30 / 15 / 14	122 / 51 / 60	64 / 22 / 32
random\randpms1000d3.xes	228 / 81 / 51	201 / 67 / 51	236 / 83 / 51	188 / 71 / 83	188 / 71 / 83	420 / 187 / 199	188 / 71 / 83	188 / 71 / 83	130 / 60 / 54	580 / 202 / 267	298 / 85 / 149
random\randpms100d1.xes	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / 28 / 29	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / 28 / 29	18 / 8 / 9
random\randpms100d2.xes	46 / 19 / 18	44 / 18 / 18	46 / 19 / 18	46 / 19 / 22	46 / 19 / 22	126 / 59 / 62	46 / 19 / 22	46 / 19 / 22	46 / 19 / 22	134 / 60 / 66	56 / 21 / 28
random\randpms100d3.xes	56 / 24 / 20	61 / 23 / 20	56 / 24 / 20	56 / 24 / 26	56 / 24 / 26	140 / 66 / 68	56 / 24 / 26	56 / 24 / 26	44 / 21 / 20	174 / 69 / 82	76 / 26 / 38
random\rands10000l20m8a10.xes	2 / 2 / 12	24 / 3 / 12	2 / 2 / 12	82 / 32 / 32	82 / 32 / 32	5860 / 54 / 816	1096 / 34 / 144	82 / 32 / 32	712 / 194 / 122	4041 / 99 / 1125	264 / 24 / 132
random\rands1000l10m4a5.xes	2 / 2 / 7	14 / 3 / 7	NA / 2 / 7	42 / 17 / 17	42 / 17 / 17	220 / 29 / 67	96 / 19 / 29	42 / 17 / 17	169 / 49 / 37	318 / 40 / 121	84 / 14 / 42
random\rands100l5m2a3.xes	2 / 2 / 5	10 / 3 / 5	2 / 2 / 5	26 / 11 / 11	26 / 11 / 11	72 / 19 / 31	42 / 13 / 17	26 / 11 / 11	57 / 19 / 17	92 / 23 / 41	40 / 10 / 20
reallife\realdocman.xes	NA / NA / NA	NA / NA / NA	NA / NA / NA	809 / 108 / 205	621 / 62 / 129	NA / NA / NA	424 / 69 / 208	809 / 108 / 205	1940 / 322 / 432	NA / NA / NA	1841 / 144 / 919
reallife\realhospital.xes	NA / NA / NA	NA / NA / NA	NA / NA / NA	3226 / 442 / 1278	3458 / 466 / 1266	22033 / 3093 / 8137	3240 / 504 / 1575	3226 / 442 / 1278	4295 / 701 / 1425	NA / NA / NA	8999 / 868 / 4499
reallife\realincman.xes	78 / 20 / 18	273 / 50 / 18	70 / 20 / 18	229 / 50 / 80	208 / 55 / 68	669 / 102 / 270	181 / 32 / 90	229 / 50 / 80	314 / 82 / 83	NA / NA / NA	310 / 34 / 155
reallife\realoutsourcing.xes	22 / 10 / 7	25 / 8 / 7	55 / 10 / 7	NA / NA / NA	98 / 33 / 28	160 / 43 / 72	64 / 14 / 32	NA / NA / NA	154 / 46 / 40	383 / 40 / 134	78 / 14 / 39
synthetic\a10skip.xes	30 / 14 / 12	30 / 14 / 12	30 / 14 / 12	30 / 13 / 13	30 / 13 / 13	78 / 38 / 38	32 / 15 / 15	30 / 13 / 13	28 / 13 / 12	86 / 39 / 42	38 / 15 / 19
synthetic\a12.xes	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	90 / 44 / 44	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	103 / 46 / 50	46 / 17 / 23
synthetic\a5.xes	18 / 8 / 7	18 / 8 / 7	NA / 8 / 7	31 / 13 / 13	31 / 13 / 13	51 / 24 / 24	25 / 11 / 11	31 / 13 / 13	25 / 11 / 11	61 / 25 / 29	34 / 12 / 17
synthetic\a6nfc.xes	18 / 9 / 8	19 / 9 / 8	18 / 9 / 8	20 / 10 / 9	20 / 10 / 9	52 / 25 / 25	24 / 10 / 12	20 / 10 / 9	20 / 10 / 9	60 / 26 / 29	24 / 10 / 12
synthetic\a7.xes	24 / 10 / 9	24 / 10 / 9	24 / 10 / 9	24 / 9 / 10	22 / 9 / 9	73 / 30 / 32	27 / 11 / 12	24 / 9 / 10	NA / NA / NA	133 / 36 / 56	54 / 15 / 27
synthetic\a8.xes	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	66 / 32 / 32	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	74 / 33 / 36	32 / 12 / 16
synthetic\betasimplified.xes	34 / 15 / 13	45 / 18 / 13	34 / 15 / 13	42 / 17 / 21	42 / 17 / 21	94 / 43 / 47	42 / 17 / 21	42 / 17 / 21	38 / 16 / 19	94 / 43 / 47	42 / 17 / 21
synthetic\choice.xes	24 / 9 / 12	24 / 9 / 12	24 / 9 / 12	48 / 18 / 24	48 / 18 / 24	96 / 42 / 48	48 / 18 / 24	48 / 18 / 24	30 / 12 / 15	96 / 42 / 48	48 / 18 / 24
synthetic\driverslicense.xes	18 / 8 / 9	22 / 10 / 9	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / 28 / 29	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / NA / NA	18 / 8 / 9
synthetic\driverslicenseloop.xes	NA / NA / NA	34 / 13 / 11	31 / 12 / 11	38 / 15 / 17	38 / 15 / 17	86 / 39 / 41	38 / 15 / 17	38 / 15 / 17	26 / 12 / 12	98 / 40 / 47	46 / 15 / 23
synthetic\herbstfig3p4.xes	26 / 12 / 12	26 / 12 / 12	26 / 12 / 12	26 / 12 / 12	26 / 12 / 12	79 / 38 / 38	28 / 13 / 13	26 / 12 / 12	26 / 12 / 12	103 / 42 / 50	50 / 17 / 25
synthetic\herbstfig5p19.xes	24 / 10 / 8	22 / 9 / 8	24 / 10 / 8	26 / 11 / 12	26 / 11 / 12	58 / 27 / 28	26 / 11 / 12	26 / 11 / 12	21 / 10 / 10	66 / 28 / 32	34 / 12 / 17
synthetic\herbstfig6p18.xes	NA / NA / NA	15 / 6 / 7	13 / 6 / 7	20 / 8 / 10	20 / 8 / 10	50 / 23 / 25	20 / 8 / 10	20 / 8 / 10	20 / 8 / 10	50 / 23 / 25	20 / 8 / 10
synthetic\herbstfig6p31.xes	18 / 5 / 9	29 / 8 / 9	18 / 5 / 9	23 / 7 / 9	18 / 5 / 9	70 / 31 / 35	18 / 5 / 9	23 / 7 / 9	18 / 5 / 9	70 / 31 / 35	18 / 5 / 9
synthetic\herbstfig6p36.xes	24 / 11 / 12	28 / 13 / 12	24 / 11 / 12	24 / 11 / 12	24 / 11 / 12	76 / 37 / 38	24 / 11 / 12	24 / 11 / 12	24 / 11 / 12	76 / 37 / 38	24 / 11 / 12
synthetic\herbstfig6p38.xes	10 / 6 / 7	14 / 7 / 7	10 / 6 / 7	26 / 11 / 9	26 / 11 / 9	58 / 23 / 27	28 / 10 / 14	26 / 11 / 9	35 / 14 / 12	64 / 24 / 29	36 / 12 / 18
synthetic\herbstfig6p41.xes	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	102 / 51 / 48	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	134 / 55 / 64	74 / 26 / 37
synthetic\l2l.xes	12 / 7 / 6	12 / 6 / 6	14 / 6 / 6	16 / 8 / 6	12 / 6 / 6	36 / 18 / 18	12 / 6 / 6	16 / 8 / 6	12 / 6 / 6	36 / 18 / 18	12 / 6 / 6
synthetic\l2loptional.xes	12 / 6 / 6	12 / 6 / 6	12 / 6 / 6	22 / 10 / 10	20 / 9 / 10	40 / 19 / 20	20 / 9 / 10	22 / 10 / 10	14 / 7 / 7	40 / 19 / 20	20 / 9 / 10
synthetic\l2lskip.xes	14 / 7 / 6	12 / 6 / 6	14 / 7 / 6	16 / 8 / 7	14 / 7 / 7	36 / 18 / 18	14 / 7 / 7	16 / 8 / 7	14 / 7 / 7	36 / 18 / 18	14 / 7 / 7
synthetic\prAm6.xes	3920 / 986 / 363	NA / NA / NA	3937 / 990 / 363	944 / 336 / 416	951 / 369 / 399	4179 / 1404 / 1842	1222 / 429 / 526	944 / 336 / 416	NA / NA / NA	11579 / 2336 / 4818	4470 / 553 / 2235
synthetic\prBm6.xes	951 / 396 / 317	1026 / 402 / 317	951 / 396 / 317	794 / 309 / 336	831 / 358 / 331	2927 / 1122 / 1291	1030 / 393 / 417	794 / 309 / 336	844 / 355 / 339	6408 / 1450 / 2615	2394 / 470 / 1197
synthetic\prCm6.xes	NA / NA / NA	NA / NA / NA	NA / NA / NA	740 / 238 / 360	781 / 275 / 350	5368 / 1328 / 2260	1210 / 408 / 542	740 / 238 / 360	1270 / 390 / 530	NA / NA / NA	NA / NA / NA
synthetic\prDm6.xes	0 / 0 / 0	NA / NA / NA	NA / NA / NA	1290 / 450 / 556	1364 / 552 / 516	13165 / 1950 / 4719	2697 / 730 / 1117	1290 / 450 / 556	1921 / 710 / 686	NA / NA / NA	NA / NA / NA
synthetic\prEm6.xes	NA / NA / NA	NA / NA / NA	3949 / 1137 / 275	761 / 293 / 310	747 / 322 / 288	4501 / 1072 / 1770	1456 / 464 / 603	761 / 293 / 310	896 / 370 / 327	NA / NA / NA	13044 / 489 / 6522
synthetic\prFm6.xes	0 / 0 / 0	NA / NA / NA	8018 / 2510 / 299	NA / 367 / 348	899 / 406 / 330	5761 / 1222 / 2204	2541 / 713 / 1020	892 / 367 / 348	1096 / 476 / 379	NA / NA / NA	NA / NA / NA
synthetic\prGm6.xes	0 / 0 / 0	NA / NA / NA	40021 / 7589 / 335	1046 / 359 / 440	1061 / 423 / 398	7620 / 1473 / 2891	2205 / 571 / 908	1046 / 359 / 440	1416 / 529 / 516	NA / NA / NA	NA / NA / NA

Table 5.5: Results of the Fodina conformance checking experiment for the Fitting Single Trace Measure metric.

Algorithm	A	A++	A6	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	PNML	PNML	PNML	HNET	HNET	FLEX	CNET
Event Log				PM[1]			
permutations\perml10a3.xes	0.625	0.489	0.940	0.360	0.180	0.430	1.000
permutations\perml3a10.xes	0.981	0.977	1.000	0.820	1.000	1.000	1.000
permutations\perml3a3.xes	0.926	0.852	1.000	0.560	1.000	1.000	1.000
permutations\perml3a5.xes	0.928	0.928	1.000	0.670	1.000	1.000	1.000
permutations\perml5a10.xes	0.902	0.909	0.985	0.690	0.970	1.000	1.000
permutations\perml5a3.xes	0.675	0.691	0.940	0.530	0.760	0.900	1.000
permutations\perml5a5.xes	0.782	0.784	0.959	0.540	0.890	0.970	1.000
random\randpms10000d1.xes	0.442	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms10000d2.xes	0.477	0.305	0.965	0.900	0.910	1.000	1.000
random\randpms10000d3.xes	0.705	0.533	0.983	0.840	0.910	0.980	1.000
random\randpms1000d1.xes	0.846	0.669	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	0.849	0.763	1.000	1.000	1.000	1.000	1.000
random\randpms1000d3.xes	0.612	0.514	0.974	0.640	0.810	0.840	1.000
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.450	0.490	0.948	0.840	0.900	0.980	1.000
random\randpms100d3.xes	0.930	0.810	1.000	1.000	1.000	1.000	1.000
random\rands10000l20m8a10.xes	0.055	0.029	0.781	0.040	0.240	0.710	1.000
random\rands1000l10m4a5.xes	0.347	0.267	0.885	0.230	0.430	0.750	1.000
random\rands100l5m2a3.xes	0.770	0.780	0.966	0.600	0.770	0.890	1.000
reallife\realdocman.xes	0.846	0.976	0.999	1.000	1.000	1.000	1.000
reallife\realhospital.xes	0.273	0.360	0.893	0.340	0.550	0.740	1.000
reallife\realincman.xes	0.800	0.951	0.995	1.000	1.000	1.000	1.000
reallife\realoutsourcing.xes	0.631	0.924	0.970	0.920	0.930	0.930	1.000
synthetic\a10skip.xes	0.467	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a12.xes	0.523	0.753	1.000	1.000	1.000	1.000	1.000
synthetic\a5.xes	0.657	0.883	1.000	0.900	1.000	1.000	1.000
synthetic\a6nfc.xes	0.800	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a7.xes	0.927	0.903	1.000	1.000	1.000	1.000	1.000
synthetic\a8.xes	0.813	0.643	1.000	1.000	1.000	1.000	1.000
synthetic\betasimplified.xes	1.000	0.493	1.000	1.000	1.000	1.000	1.000
synthetic\choice.xes	0.950	0.873	1.000	0.920	1.000	1.000	1.000
synthetic\driverslicense.xes	0.500	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicenseloop.xes	0.483	0.654	0.953	0.670	0.810	0.780	1.000
synthetic\herbstfig3p4.xes	0.906	0.781	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig5p19.xes	0.883	0.767	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p18.xes	0.283	0.517	0.878	0.750	1.000	0.860	1.000
synthetic\herbstfig6p31.xes	0.293	0.770	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p36.xes	0.450	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p38.xes	0.850	0.857	1.000	0.560	1.000	1.000	1.000
synthetic\herbstfig6p41.xes	0.750	0.750	1.000	1.000	1.000	1.000	1.000
synthetic\l2l.xes	1.000	1.000	1.000	0.750	1.000	1.000	1.000
synthetic\l2loptional.xes	0.237	0.997	0.972	1.000	1.000	1.000	1.000
synthetic\l2lskip.xes	0.490	1.000	0.944	1.000	1.000	1.000	1.000
synthetic\prAm6.xes	0.959	0.965	1.000	1.000	1.000	1.000	1.000
synthetic\prBm6.xes	0.941	0.959	1.000	1.000	1.000	1.000	1.000
synthetic\prCm6.xes	0.954	0.976	1.000	1.000	1.000	1.000	1.000
synthetic\prDm6.xes	0.968	0.981	1.000	1.000	1.000	1.000	1.000
synthetic\prEm6.xes	0.962	0.971	1.000	0.990	1.000	1.000	1.000
synthetic\prFm6.xes	0.963	0.976	1.000	1.000	1.000	1.000	1.000
synthetic\prGm6.xes	0.968	0.970	1.000	1.000	1.000	1.000	1.000

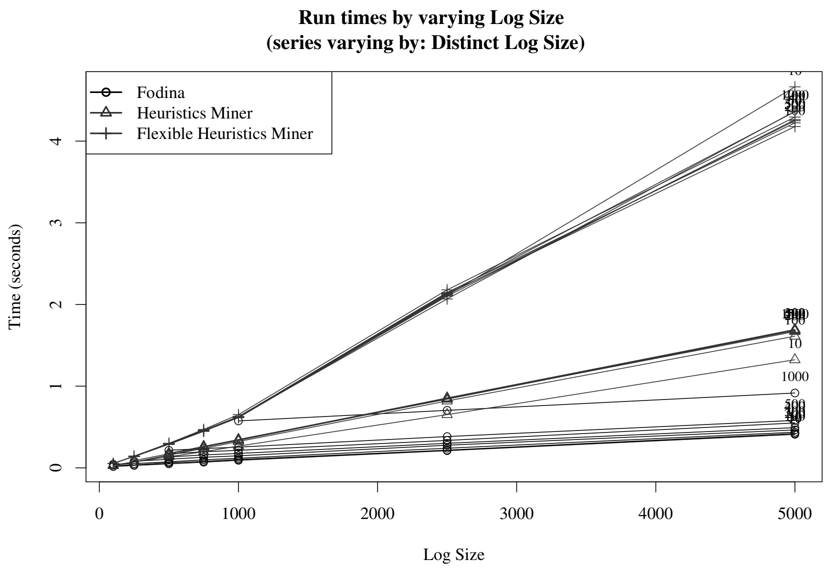


Figure 5.9: Scalability results for Fodina compared to (Flexible) Heuristics Miner (ProM 6). The effects of total log size, distinct log size, number of activities and trace length were investigated. The results show that Fodina performs faster than comparable technique. This figure shows run times as depending on log size and distinct log size.

similar complexity characteristics as other heuristic miners (non-linearity in the number of activities), the concurrent discovery of semantic split/join information greatly speeds up the discovery procedure. In addition, the other tested miners suffer from implementation drawbacks, most notably stemming from the fact that multiple passes over the event log are performed.

This experimental section has illustrated the quality, robustness and scalability of our proposed technique. The next section shows how Fodina can be combined with other discovery perspectives towards the discovery of bidimensional process models.

5.6 Application: Bidimensional Process Discovery with BPMN

In this section, we explore the possibilities of applying Fodina towards so-called “bidimensional process discovery”, where the perspective of control-flow is com-

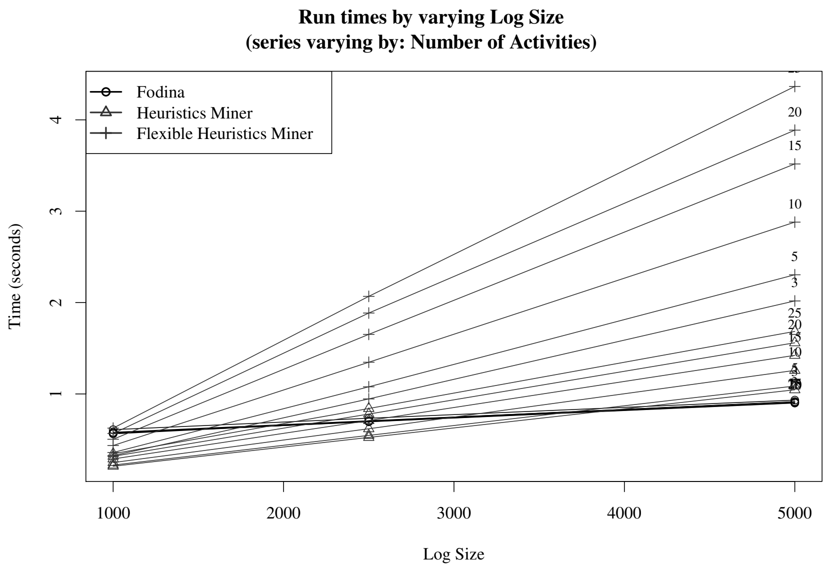


Figure 5.9 (continued): This figure shows run times as depending on log size and number of activities.

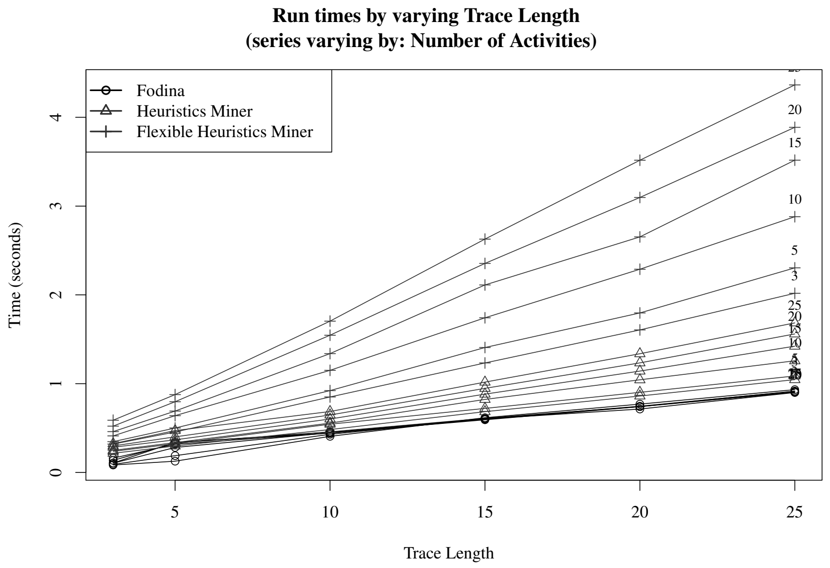


Figure 5.9 (continued): This figure shows run times as depending on trace length and number of activities.

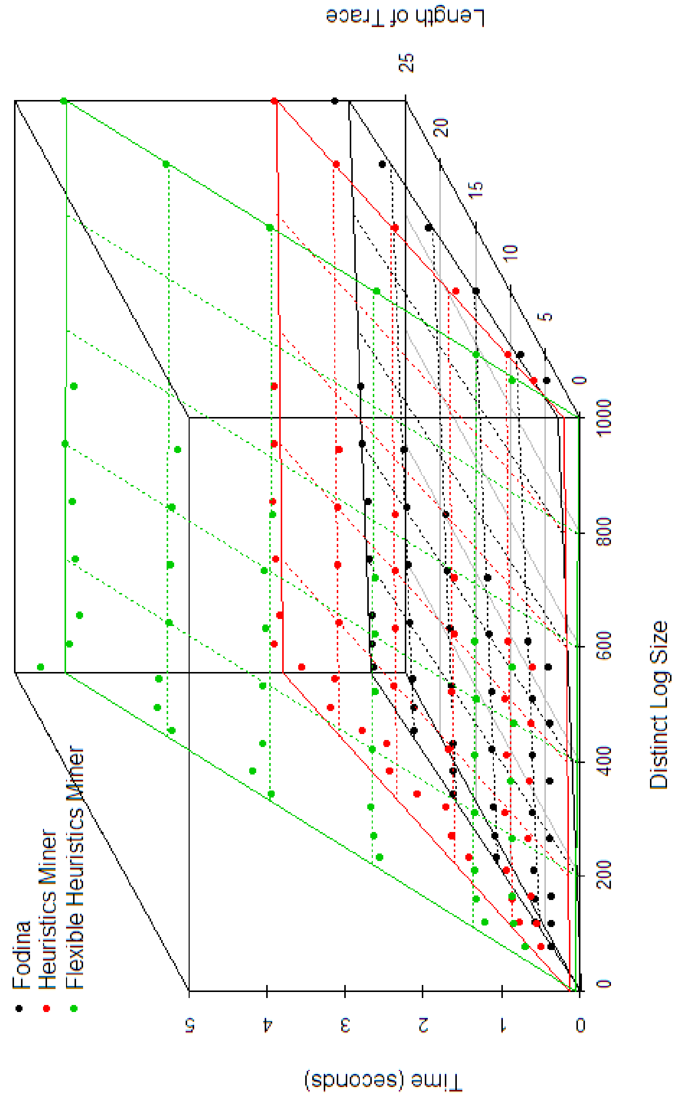


Figure 5.9 (continued): Three dimensional aggregated summary plot comparing Fodina (lower plane), Heuristics Miner (middle plane) and Flexible Heuristics Miner (top plane).

bined with other context, such as—as will be developed here—the organizational context of a process. In addition, we will depict the discovered process model in the form of a BPMN model. As we have seen, popular output formats for discovered models include: Petri nets [38], Heuristic nets [44], Causal nets [43] and EPC's [146]. However, one particular process modeling standard which has been somewhat overlooked in the process mining community is the Business Process Model and Notation (BPMN) standard [161]. This is peculiar, as the majority of educators and researchers have adopted BPMN as the language of choice when working with business processes. The reason for this stems mainly from the fact that BPMN has, for a long time, lacked a formal definition of its execution semantics. The initial specifications [162] defined behavioral semantics using the notion of token flow, similar to Petri nets and UML activity diagrams, but solely described the execution semantics in narrative form. Although researchers have defined formalized definitions, ranging from attempts to define a formal semantics for a subset of BPMN [163, 164] to more complete approaches [163, 165, 166], the fact remains that both BPMN's many visual objects and its weak semantic formalization have caused scholars to develop process discovery techniques based on more formalized modeling approaches.

Nevertheless, given BPMN's wide dissemination, we argue that the availability of a native BPMN-based process discovery technique could be of great benefit within process identification, optimization and re-engineering efforts. Therefore, this section will opt to represent discovered control-flow aspects using BPMN constructs, with Fodina as the underlying miner. We select a subset of constructs, both because discovering some constructs is near-impossible using only historically recorded process execution "traces", as well as because scholars have indicated that only a small subset of BPMN's constructs are used by the majority of practitioners [167]. However, our proposed approach is unique in the sense that it combines the control-flow perspective with an organizational dimension by discovering swim lanes that represent organizational roles in the business process. In addition, our technique provides intuitive and easy-to-use abstraction/specification functionality which makes it applicable to event logs with various complexity levels, provides instant feedback about the conformance between the input log and the resulting model based on a dedicated fitness metric, is robust to noise, and can easily be integrated with modeling and other BPM tools by exporting the discovered model.

5.6.1 Rationale

5.6.1.1 Relevance for Practitioners

BPMN is considered as the de facto standard for process modeling [168] and is widely adopted by both business and IT communities [169]. Practitioners from both communities use the notation standard mainly for documenting, improving, simulating and implementing business processes [170]. While the adoption of BPMN for the purpose of process modeling has been successful, the adoption of process mining as the most valuable tool for business process improvement initiatives is somehow lagging. It is argued that a core factor contributing to this effect consists of a lack of deep technical understanding from typical business practitioners involved in such improvement initiatives with regard to conventional languages used by process mining tools. Even despite the uptake of commercial and highly user-friendly process discovery tools, the mismatch in modeling notation and the subsequent translation effort required to go from the analysis phase to redesign brings about an unnecessary adoption barrier. Therefore, we contend that the mining of BPMN models from execution data will prove highly beneficial for the further adoption of process mining, along the following lines of reasoning:

Automated Process Identification Practitioners involved in process identification and modeling, can be persuaded into using automated process discovery techniques if such techniques provide effortless integration with popular modeling tools. With the capability to discover BPMN models from event logs, actual time savings can be realized for practitioners who are now typically involved in a two-step process of first interpreting automated discovery results and then making use of the insights gained for designing or adapting process models. Note that the survey in [154] showed that process model editing functionality is amongst the most desired additional features for the ProM-framework. This clearly indicates that (automated) analysis and (re)design are tightly coupled and tools and techniques in both areas should be maximally aligned.

Facilitating the Process Re-engineering Cycle In typical redesign scenarios, people observe the as-is state of a process or set of processes, take certain improvement decisions, analyze the outcomes, and subsequently take additional improvement measures if necessary. Currently available automated process discovery techniques require business (process) analysts to possess additional skills and knowledge about typical output modeling notations such as Petri nets, Heuristic nets, or Fuzzy models. In addition, next to interpretation, practitioners will also be required to compare the results of automated discovery with existing process models. Such a comparison is far from effortless requiring profound technical understanding often unavailable in organizations with lower BPM maturity. With discovered process models in native BPMN format, the mapping of discovered vs. existing models becomes significantly easier.

Improved Communication of Process Mining Results Working with a unified process model notation throughout the entire BPM life cycle will enable improved communication between functional units as well as across organizational hierarchies. Due to the fact that many organizations heavily rely on process modeling for documentation and communication, investments in data collection and data analysis might be perceived more worthwhile because these investments should not be looked at in isolation, but can actually contribute to and improve existing BPM practices.

5.6.1.2 Relevance for Education

A second stakeholder group for which BPMN-based process discovery is of value is educators and students. Generally speaking, BPM courses and text books (e.g. [171]) kick off with a thorough discussion on process modeling, with BPMN often receiving a great deal of attention. In later stages or chapters, process mining is brought up as well. However, this often requires educators to introduce new modeling notations, most notably Petri nets, Heuristic nets, and Fuzzy models, given their popularity for process mining. Moreover, the introduction of such new paradigms quickly obfuscates the link with process modeling and execution topics. While several programs can already leverage upon previously acquired knowledge, a majority of students, e.g. in business/management-oriented studies, do not possess such background knowledge. For that reason, a high-quality process discovery tool which presents its results in BPMN is likely to lower the effort required by educators to incorporate process mining in their units. It is pointed out that, from a student perspective, a positive attitude towards the usability and ease of use has been observed with respect to BPMN and its tool support [172].

5.6.1.3 Relevance for Research

Key research contributions in the process mining field have traditionally been strongly technical in nature. While valuable application studies have been published as well, there exists a significant opportunity for research about topics such as usefulness, ease of use, user acceptance, etc. of process mining within organizations. A process discovery technique with BPMN as underlying modeling language will lower barriers to conducting such studies, which often involve technically lower skilled individuals. Ultimately, user-centered studies could provide valuable insights into how the full potential of process mining can be realized or in what directions future process mining research should develop.

5.6.2 Comparative Study

The quality of discovered process models is inherently determined by the implicit search space implied by the representational bias of a process discovery technique (and thus its associated representation language). In [173], the authors advocate for selecting the "right" representational bias when discovering process models from event logs. They argue that the representational bias should be based on essential properties of a process model and should not be driven by the desired graphical representation. The process mining manifesto also lists the aspect of representational bias as one of the key challenges in the process mining domain [174].

While we don't contest that the search for an optimal representational bias for process discovery in terms of the implicit search space is of interest, a more pragmatic stance is taken in this work. This is because, from a knowledge discovery viewpoint [20, 22], patterns discovered from data should adhere to several principles: validity, novelty, usefulness, and understandability. While the use of for instance Causal nets for process discovery is likely to produce rich and high quality results, such an approach suffers from a steep learning curve which often leads to problems of understandability. For this reason, we argue that a more pragmatic, user-centered stance with respect to the suitability of the representational bias is worth pursuing. This pragmatic approach is based upon an assessment of some key characteristics of process modeling notations for the purpose of process discovery, as detailed in Table 5.6. It is argued that there exist two contrasting effects that make it difficult to agree on one fit-for-all modeling notation for process discovery.

Table 5.6: Key characteristics of process modeling notations for the purpose of process discovery.

Modeling Notation	Ease of Interpretation	Suitability Rep. Bias Proc. Disc.	Popularity (Modeling)	Popularity (Mining)
Petri net	●●○○○	●●○○○	●○○○○	●●●●○
Heuristic net	●●●○○	●●●●○	○○○○○	●●●●●
Fuzzy model	●●●●○	●●●●○	○○○○○	●●●●●
Causal net	●○○○○	●●●●●	○○○○○	●●○○○
EPC	●●●●○	●○○○○	●●●●○	●●○○○
BPMN	●●●●●	●●○○○	●●●●●	○○○○○

Based on a comparative analysis of various modeling notations, it is observed that traditionally popular modeling notations used for process discovery (i.e. Heuristic nets, Fuzzy models, and Causal nets) put a strong emphasis on the suitability of the representational bias. Note that our judgment about the representational bias reflects how well these notations help process discovery techniques at expressing a large number of possible constructs, while at the same time avoiding syntactically incorrect models as much as possible. Therefore, Petri nets, another popular representation choice, are scored lower because it is actually non-trivial to avoid the construction of incorrect Petri nets. On the other hand, representation languages with a less steep learning curve such as EPC and BPMN make it more difficult for process discovery techniques in terms of representational bias because modeling constructs are difficult to map against recorded data and because of the broad set of available constructs in the case of BPMN. A second, even stronger, contrast exists in terms of the level of popularity for modeling vs. mining. Basically, there exists an important discrepancy in the BPM domain between languages used for modeling and languages used for mining. While some might argue that models can be translated from one language to another (for instance through the use of Petri nets as BPM's Esperanto), this often involves non-trivial procedures. We opt to bridge the gap between modeling and mining by making use of BPMN as the representational language.

To conclude, we recognize that BPMN presents several drawbacks as a representational language for process discovery. Most importantly, many of its concepts are difficult or impossible to map with recorded event data. In addition, the broad range of concepts also leads to the absence of a clear and crisp definition of its execution semantics, which is a much desired characteristic for process discovery. However, given the rationale in Section 5.6.1 for a native BPMN miner, the next section details how these limitations can be dealt with.

5.6.3 Implementation

We deliberately considers a subset of BPMN's notational constructs in order to perform the control-flow discovery. Other works have illustrated that only a small subset of BPMN is actually being applied in real-life modeling practice [167], those being gateways (XOR and parallel), tasks, sequence flow, start and end event, and swim lanes. All of these constructs are also supported by our approach. In addition, we highlight the fact that discovered BPMN process models by our approach do provide an ideal starting point which can easily be adapted, extended, and modified by modelers and practitioners, as the discovered model lies much closer to the representational language practitioners are already applying, thus preventing conversion steps (with potential loss of accuracy or behavioral representation) or having to learn other modeling notations.

To perform the control-flow discovery, we first apply Fodina in order to derive a Causal net. Based on this information, a BPMN model is constructed as follows. First, start and end events are added. Second, the BPMN graph is constructed. Activities are added for each $a \in A_L$: A_a. Next, a XOR input and output gateway is created for each activity A_a: XOR^i_a and XOR^o_a and connected to the activity with sequence flows (XOR^i_a, A_a) and (A_a, XOR^o_a). For each $a \in A_L$, AND gateways are created for each set of activities in $I(a)$ and $O(a)$: $AND^{i,j}_a$ and $AND^{o,k}_a$, which are connected to the input and output XOR gateways with sequence flows $(AND^{i,j}_a, XOR^i_a)$ for $j = 1, \ldots, |I(a)|$ and $(XOR^o_a, AND^{o,k}_a)$ for $k = 1, \ldots, |O(a)|$. Next, a set of XOR connecting gateways is constructed for each $a_i \in \bigcup(I(a))$ with $a \in A_L, a \neq a_i$: $XOR^{a_i,a}_d$ and for each $a_o \in \bigcup(O(a))$ with $a \in A_L, a \neq a_o$: XOR^{a,a_o}_d. Sequence flows are added for each $(AND^{o,k}_a, XOR^{a_i,a_o}_d)$ so that $a_i \in A_L, a_o \in A_L, a \in A_L, k = 1, \ldots |O(a)|$ and with $a_i \in O(a)^k$[9]. Similarly, we add flows for each $(XOR^{a_i,a_o}_d, AND^{i,j}_a)$ so that $a_i \in A_L, a_o \in A_L, a \in A_L, j = 1, \ldots |I(a)|$ and with $a_o \in I(a)^j$. Finally, the start and end event are connected with all the activities A_a for which $I(a) = \{\emptyset\}$ and $O(a) = \{\emptyset\}$ respectively. If no such activity can be found, the algorithm determines a single start/end activity based on the frequency of the activity occurring most commonly at the start/end of traces. On the other hand, if multiple start or ending activities can be identified, they are connected through a XOR gateway with the starting and ending event. The third phase of the control-flow discovery algorithm consists of simplification of the BPMN model. This simplification step consists of iterative removal of all gateways which only contain a single incom-

[9] We assume here that the input and output sets are ordered. $O(a)^k$ thus returns the jth subset of $O(a)$.

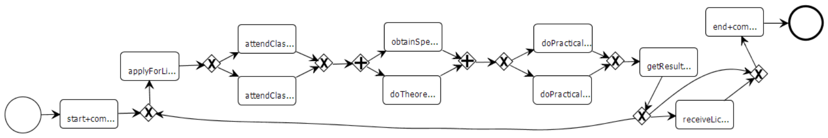
ing and one outgoing sequence flow, merging all AND gateways with the same incoming activities and a single outgoing activity (using a XOR gateway to connect the merged outgoing activities to the AND gateway) and merging all AND gateways with the same outgoing activities and a single incoming activity (using a XOR gateway to connect the merged incoming activities to the AND gateway).

The majority of existing process discovery algorithms focus on discovering the control-flow perspective of an event log, meaning that they use the sequence and ordering of activities to derive a process model using a particular representational language. Other techniques exist which start from another event log perspective (social network extraction techniques [175], for instance). Here, we apply a bidimensional approach, directly incorporating the social (i.e. originator) perspective in the discovered model, together with control-flow (i.e. the sequence flow between activities).

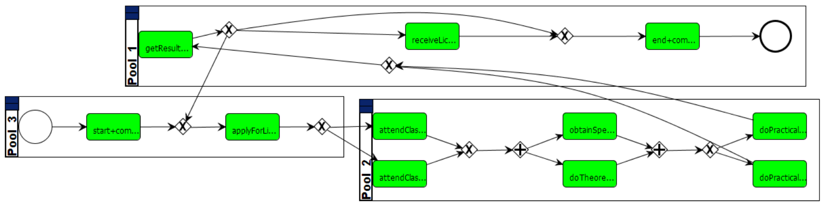
To model originator information, our technique makes use of the swimlane construct of BPMN, meaning that our technique attempts to create a number of swimlanes containing one or more activities. Each swimlane then represent a “worker pool” (or “role”) which is responsible for executing its contained activities. The swimlanes are discovered as follows. Recall $o : L \to O_L$ as being the function returning the originator (i.e. the role, person, group, department...) having executed an event. Depending on the desired level of granularity, end-users can choose which originator attribute to use to construct the swimlanes. First, for each activity $a \in A_L$, a dedicated swimlane-representing set $S_i = \{a\}$ is constructed, containing this single activity, so that $S = \{S_1, \ldots, S_{|T_L|}\}$. Next, swimlanes S_i and S_j are merged iff $\exists o \in O_L, a \in S_i, b \in S_j, \sigma \in L, \tau \in L : [i \neq j \wedge a(\sigma) = a \wedge a(\tau) = b \wedge o(\sigma) = o(\tau) = o]$. Ultimately, this leads to a set of merged swimlanes representing a grouping of activities which are to be contained in their swimlane. The grouping is performed such that each swimlane also represents a “role” (a distinct grouping of originators) responsible for this set of activities.

5.6.4 Illustrating Example

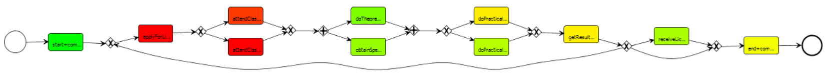
This section illustrates the developed miner by means of a concise, fictitious example. To do so, we utilize the *driversLicenseLoop* event log (a commonly used log in benchmarking setups, see [65]), and annotate this event log with originator information. Figure 5.10 depicts a number of screen captions illustrating the



(a) Mined BPMN model without swimlanes or conformance visualization.



(b) Mined BPMN model with swimlanes and conformance visualization.



(c) Conformance analysis of mined BPMN diagram against noisy event log.

Figure 5.10: Screen captions illustrating discovery features of BPMN Miner.

features of the BPMN Miner plugin. Figure 5.10(a) shows the result of mining the BPMN model from the *driversLicenseLoop* event log without creating swimlanes or performing conformance analysis. This model can be exported to XPDL and imported in third-party tools or converted in ProM to a Petri net and used for further analysis. Next, Figure 5.10(b) shows the discovered BPMN model with the bidimensional discovery and conformance analysis enabled. As shown, the model has a perfect fitness level (all activities green). Each swimlane represents a pool of originators responsible for the activities contained within the swimlane. Figure 5.10(c) shows the result of a conformance analysis of the mined BPMN diagram (without swimlanes) against the event log in which a high amount of noise was introduced and does thus not fit the discovered model. The coloring of activities (green to red scale) indicate problematic areas with a high amount of deviations.

5.7 Conclusions

This chapter has presented Fodina, a process discovery technique which follows the generic idea of heuristic process discovery algorithms. Although such techniques have proven themselves as robust process discovery algorithms and able to deal with real-life event logs containing a large amount of variety of behavior, we identified some particular issues which limit the robustness and reliability of the technique. As such, we have set out to perform a thorough literature review and evaluation of the existing heuristic process discovery variants with their implementation to consequently propose a new technique which was proven to be more robust via a comprehensive evaluation experiment. Furthermore, the proposed technique puts forward some novel contributions, most notably the capability to mine duplicate tasks and the ability to configure various options to guide the discovery algorithms towards particular process model solutions.

Concerning future work, we plan to expand the feature set of Fodina, particularly concerning duplicate task mining, by allowing users to provide feedback concerning which duplicate tasks should be merged, thus allowing the possibility to "collapse" the process model and inspect the resulting impact on fitness and structural clarity. Another interesting opportunity is to separate the method for duplicate tasks mining in a dedicated event log "prefilterer", so that the technique can also be applied to enable other discovery algorithm to discover duplicate activities in this manner. It is also possible to devise more advanced techniques to "duplicate" tasks, for example based on frequent item sets [176–179].

Finally, an interesting avenue for further work is to combine the aspect of process discovery with (artificial) negative events. Recall for instance that we have argued that the direct succession of a task $\mathtt{a}$ after $\mathtt{b}$ is not always suitable direct counter-evidence against the direct succession of $\mathtt{b}$ after $\mathtt{a}$. However, what would be suitable counter-evidence would be the occurrence of a negative event $\mathtt{b}^-$ after $\mathtt{a}$. As such, changing the dependency metrics to take into account negative information seems like a straightforward and interesting possibility for future research.

5.8 Experimental Result Tables

The following tables present the full set of results obtained in the Fodina benchmarking experiment (Section 5.5).

Table 5.7: Results of the Fodina conformance checking experiment. This table lists the results for the Average Alignment Based Trace Fitness metric.

Algorithm	A	A++	A6	HM5	HM6	FHM6	F	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	PNML	PNML	PNML	PNML	PNML	FLEX	PNML	PNML	PNML	FLEX	PNML
Event Log						f_a^{avg}					
permutations\perml10a3.xes	0.000	1.000	NA	NA	NA	1.000	NA	NA	NA	1.000	NA
permutations\perml3a10.xes	0.000	1.000	NA	NA	NA	1.000	0.999	NA	NA	1.000	1.000
permutations\perml3a3.xes	NA	1.000	NA	0.750	0.822	1.000	0.889	0.750	0.901	1.000	1.000
permutations\perml3a5.xes	NA	1.000	NA	0.787	0.787	1.000	0.996	0.787	0.787	1.000	1.000
permutations\perml5a10.xes	0.000	1.000	NA	NA	NA	NA	NA	NA	NA	1.000	NA
permutations\perml5a3.xes	0.000	NA	NA	NA	NA	1.000	NA	NA	NA	1.000	NA
permutations\perml5a5.xes	0.000	NA	NA	NA	NA	1.000	NA	NA	NA	1.000	NA
random\randpms10000d1.xes	0.000	NA	NA	NA	NA	1.000	NA	NA	NA	1.000	NA
random\randpms10000d2.xes	0.000	NA	NA	NA	NA	1.000	NA	NA	NA	1.000	NA
random\randpms10000d3.xes	0.000	NA	NA	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d1.xes	NA	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	0.353	0.830	0.353	1.000	1.000	1.000	1.000	1.000	0.868	1.000	1.000
random\randpms1000d3.xes	0.115	0.295	0.115	1.000	1.000	1.000	1.000	1.000	0.727	1.000	1.000
random\randpms100d1.xes	NA	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.230	0.700	0.230	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d3.xes	0.558	0.853	0.558	1.000	1.000	1.000	1.000	1.000	0.928	1.000	1.000
random\rands10000l20m8a10.xes	NA	1.000	NA	NA	NA	NA	NA	NA	NA	1.000	1.000
random\rands1000l10m4a5.xes	NA	1.000	NA	0.961	0.961	1.000	0.992	0.961	0.664	1.000	1.000
random\rands100l5m2a3.xes	NA	1.000	NA	0.947	0.947	1.000	0.982	0.947	0.520	1.000	1.000
reallife\realdocman.xes	NA	NA	NA	NA	NA	NA	0.975	NA	NA	NA	0.996
reallife\realhospital.xes	NA	NA	NA	NA	NA	NA	0.857	NA	NA	NA	NA
reallife\realincman.xes	NA	NA	NA	NA	NA	1.000	0.977	NA	0.604	1.000	0.999
reallife\realoutsourcing.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\a10skip.xes	0.972	0.972	0.972	1.000	1.000	1.000	1.000	1.000	0.972	1.000	1.000
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a5.xes	1.000	1.000	NA	1.000	1.000	1.000	1.000	1.000	0.943	1.000	1.000
synthetic\a6nfc.xes	0.985	1.000	0.985	0.985	0.985	1.000	0.995	0.985	0.985	1.000	0.995
synthetic\a7.xes	1.000	1.000	1.000	0.974	0.943	1.000	0.934	0.974	NA	1.000	0.996
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\betasimplified.xes	0.725	0.851	0.725	1.000	1.000	1.000	1.000	1.000	0.920	1.000	1.000
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.942	1.000	1.000
synthetic\driverslicense.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicenseloop.xes	NA	0.995	0.153	1.000	1.000	1.000	1.000	1.000	0.738	1.000	1.000
synthetic\herbstfig3p4.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig5p19.xes	0.707	0.869	0.707	1.000	1.000	1.000	1.000	1.000	0.644	1.000	1.000
synthetic\herbstfig6p18.xes	NA	NA	NA	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p31.xes	1.000	1.000	1.000	0.778	1.000	1.000	1.000	0.778	1.000	1.000	1.000
synthetic\herbstfig6p36.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p38.xes	NA	NA	NA	0.799	0.799	1.000	0.977	0.799	0.545	1.000	1.000
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\l2l.xes	1.000	1.000	1.000	0.851	1.000	1.000	1.000	0.851	1.000	1.000	1.000
synthetic\l2loptional.xes	1.000	1.000	1.000	0.928	1.000	1.000	1.000	0.928	0.901	1.000	1.000
synthetic\l2lskip.xes	0.427	1.000	0.427	0.875	1.000	1.000	1.000	0.875	1.000	1.000	1.000
synthetic\prAm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.999
synthetic\prBm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	0.925	NA	0.999
synthetic\prCm6.xes	NA	NA	NA	0.000	0.000	NA	0.395	0.000	0.000	NA	NA
synthetic\prDm6.xes	0.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prEm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prFm6.xes	0.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	0.000	NA	0.102	NA	NA	NA	NA	NA	NA	NA	NA

Table 5.8: Results of the Fodina conformance checking experiment. This table lists the results for the One Align Precision metric.

Algorithm	A	A++	A6	HM5	HM6	FHM6	F	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						a_p^1					
permutations\perml10a3.xes	NA	0.711	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml3a10.xes	NA	0.711	NA	NA	NA	0.868	0.917	NA	NA	0.769	0.750
permutations\perml3a3.xes	NA	0.647	NA	1.000	1.000	0.533	0.923	1.000	0.435	0.792	0.750
permutations\perml3a5.xes	NA	0.680	NA	0.444	0.440	0.621	0.703	0.444	0.440	0.781	0.750
permutations\perml5a10.xes	NA	0.776	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	0.926	0.926	0.844	0.926	0.926	0.926	NA	0.838
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	1.000	0.682	1.000	0.815	0.815	0.830	0.815	0.815	1.000	0.510	0.781
random\randpms1000d3.xes	1.000	0.803	1.000	0.695	0.695	0.650	0.695	0.695	0.844	0.186	0.632
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	1.000	0.829	1.000	0.851	0.851	0.881	0.851	0.851	0.851	0.481	0.820
random\randpms100d3.xes	1.000	0.770	1.000	0.886	0.886	0.810	0.886	0.886	0.983	0.349	0.795
random\rands10000l20m8a10.xes	NA	0.243	NA	NA	NA	NA	NA	NA	NA	NA	0.307
random\rands1000l10m4a5.xes	NA	0.436	NA	0.266	0.266	0.281	0.285	0.266	0.225	NA	0.523
random\rands100l5m2a3.xes	NA	0.631	NA	0.472	0.473	0.447	0.486	0.472	0.635	0.382	0.727
reallife\realdocman.xes	NA	NA	NA	NA	NA	NA	0.890	NA	NA	NA	0.711
reallife\realhospital.xes	NA	NA	NA	0.036	0.044	NA	0.099	0.036	0.052	NA	NA
reallife\realincman.xes	0.508	0.000	0.381	NA	NA	NA	0.955	NA	0.818	NA	0.811
reallife\realoutsourcing.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\a10skip.xes	1.000	1.000	1.000	1.000	1.000	0.915	1.000	1.000	1.000	0.609	0.918
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	0.878	1.000	1.000	1.000	0.527	0.870
synthetic\a5.xes	0.999	0.999	0.500	0.929	0.929	0.840	0.999	0.929	0.931	0.528	0.876
synthetic\a6nfc.xes	0.873	1.000	0.873	0.861	0.861	0.797	0.917	0.861	0.861	0.488	0.917
synthetic\a7.xes	1.000	1.000	1.000	0.998	0.881	0.799	1.000	0.998	NA	0.321	0.805
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	0.932	1.000	1.000	1.000	0.694	0.933
synthetic\betasimplified.xes	1.000	0.636	1.000	0.857	0.857	0.923	0.857	0.857	0.898	0.923	0.857
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicense.xes	0.889	1.000	1.000	0.889	0.889	0.952	0.889	0.889	0.889	0.952	0.889
synthetic\driverslicenseloop.xes	NA	0.931	1.000	0.646	0.646	0.709	0.646	0.646	0.855	0.325	0.733
synthetic\herbstfig3p4.xes	0.967	0.967	0.967	0.967	0.967	0.883	0.961	0.967	0.967	0.628	0.848
synthetic\herbstfig5p19.xes	1.000	1.000	1.000	0.878	0.878	0.822	0.878	0.878	1.000	0.632	0.821
synthetic\herbstfig6p18.xes	NA	0.000	0.499	0.926	0.926	0.963	0.926	0.926	0.926	0.963	0.926
synthetic\herbstfig6p31.xes	0.563	1.000	0.563	0.556	0.563	0.731	0.563	0.556	0.563	0.731	0.563
synthetic\herbstfig6p36.xes	0.917	1.000	0.917	0.917	0.917	0.967	0.917	0.917	0.917	0.967	0.917
synthetic\herbstfig6p38.xes	0.238	0.437	0.238	0.745	0.745	0.328	0.701	0.745	1.000	0.323	0.665
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	0.781	1.000	1.000	1.000	0.470	0.804
synthetic\l2l.xes	0.999	0.998	0.999	0.833	0.999	0.999	0.999	0.833	0.999	0.999	0.999
synthetic\l2loptional.xes	1.000	1.000	1.000	0.937	1.000	1.000	1.000	0.937	1.000	1.000	1.000
synthetic\l2lskip.xes	1.000	0.905	1.000	0.857	0.998	0.999	0.998	0.857	0.998	0.999	0.998
synthetic\prAm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.397
synthetic\prBm6.xes	NA	NA	NA	NA	NA	NA	NA	0.553	0.514	NA	0.457
synthetic\prCm6.xes	NA	NA	NA	NA	NA	NA	0.370	NA	NA	NA	NA
synthetic\prDm6.xes	NA	NA	NA	NA	0.164	NA	NA	0.159	0.152	NA	NA
synthetic\prEm6.xes	NA	NA	0.250	0.126	0.140	NA	0.180	NA	0.148	NA	NA
synthetic\prFm6.xes	NA	NA	0.371	NA	0.109	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	NA	NA	NA	0.160	0.180	NA	0.057	0.160	0.263	NA	NA

Table 5.9: Results of the Fodina conformance checking experiment. This table lists the results for the Best Align Precision metric.

Algorithm	A	$A++$	$A6$	HM5	HM6	FHM6	F	$HM5_L$	$HM6_L$	$FHM6_L$	F_L
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						a_p					
permutations\perml10a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml3a10.xes	0.579	0.713	0.579	0.667	0.666	0.863	0.938	0.667	0.666	0.898	0.896
permutations\perml3a3.xes	0.531	0.654	0.531	1.000	1.000	0.640	0.988	1.000	0.642	0.902	0.864
permutations\perml3a5.xes	0.555	0.684	0.555	0.671	0.670	0.690	0.787	0.671	0.670	0.920	0.893
permutations\perml5a10.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	0.933	0.933	NA	0.933	0.933	0.933	NA	0.846
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	1.000	0.923	1.000	0.874	0.872	0.826	0.873	0.874	1.000	0.610	0.833
random\randpms1000d3.xes	1.000	0.932	1.000	0.709	0.710	NA	0.710	0.709	0.843	0.187	0.641
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	1.000	0.923	1.000	0.863	0.863	0.889	0.863	0.863	0.863	0.551	0.846
random\randpms100d3.xes	1.000	0.826	1.000	0.923	0.925	0.819	0.923	0.923	0.993	0.424	0.827
random\rands10000l20m8a10.xes	0.242	0.244	0.242	NA	NA	NA	NA	NA	NA	NA	0.333
random\rands1000l10m4a5.xes	0.399	0.439	0.399	0.324	0.323	NA	0.336	0.324	NA	NA	0.577
random\rands100l5m2a3.xes	0.526	0.636	0.526	0.613	0.604	0.560	0.633	0.613	0.987	0.438	0.818
reallife\realdocman.xes	NA	NA	NA	NA	NA	NA	0.940	NA	NA	NA	NA
reallife\realhospital.xes	NA	NA	NA	NA	NA	NA	0.113	NA	NA	NA	NA
reallife\realincman.xes	0.773	0.998	0.737	NA	NA	NA	0.984	NA	NA	NA	0.908
reallife\realoutsourcing.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\a10skip.xes	1.000	1.000	1.000	1.000	1.000	0.907	1.000	1.000	1.000	0.749	0.944
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	0.878	1.000	1.000	1.000	0.720	0.897
synthetic\a5.xes	1.000	1.000	0.838	0.962	0.955	0.873	1.000	0.962	1.000	0.729	0.932
synthetic\a6nfc.xes	0.984	1.000	0.984	0.996	0.996	0.799	0.943	0.996	0.996	0.688	0.943
synthetic\a7.xes	1.000	1.000	1.000	0.997	0.906	0.705	1.000	0.997	NA	0.445	0.882
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	0.919	1.000	1.000	1.000	0.836	0.959
synthetic\betasimplified.xes	1.000	0.773	1.000	0.885	0.885	0.943	0.885	0.885	0.932	0.943	0.885
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicense.xes	0.907	1.000	0.907	0.907	0.907	0.969	0.907	0.907	0.907	NA	0.907
synthetic\driverslicenseloop.xes	NA	0.942	1.000	0.719	0.720	0.741	0.719	0.719	0.905	0.397	0.800
synthetic\herbstfig3p4.xes	0.977	0.977	0.977	0.977	0.977	0.909	0.969	0.977	0.977	0.776	0.899
synthetic\herbstfig5p19.xes	1.000	1.000	1.000	0.957	0.974	0.900	0.974	0.957	1.000	0.785	0.945
synthetic\herbstfig6p18.xes	NA	0.999	0.816	0.951	0.951	0.973	0.951	0.951	0.951	0.973	0.951
synthetic\herbstfig6p31.xes	0.589	1.000	0.589	1.000	0.589	0.853	0.589	1.000	0.589	0.853	0.589
synthetic\herbstfig6p36.xes	0.953	1.000	0.953	0.953	0.953	0.996	0.953	0.953	0.953	0.996	0.953
synthetic\herbstfig6p38.xes	0.493	0.581	0.493	0.811	0.811	0.436	0.796	0.811	NA	0.387	0.739
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	0.803	1.000	1.000	1.000	0.412	0.822
synthetic\l2l.xes	1.000	0.999	0.999	1.000	0.999	0.999	0.999	1.000	0.999	0.999	0.999
synthetic\l2loptional.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\l2lskip.xes	1.000	0.904	1.000	1.000	0.999	0.999	0.999	1.000	0.999	0.999	0.999
synthetic\prAm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prBm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	0.462
synthetic\prCm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prDm6.xes	NA	NA	NA	0.180	0.152	NA	NA	0.180	0.162	NA	NA
synthetic\prEm6.xes	NA	NA	0.243	NA	NA	0.316	0.211	0.216	NA	NA	NA
synthetic\prFm6.xes	NA	NA	0.331	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	NA	NA	NA	0.145	0.176	NA	0.082	NA	0.141	NA	NA

Table 5.10: Results of the Fodina conformance checking experiment. This table lists the results for the ETC Precision metric.

Algorithm	α	$\alpha++$	$\alpha\#$	HM5	HM6	FHM6	F	$HM5_L$	$HM6_L$	$FHM6_L$	F_L
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						etc_p					
permutations\perml10a3.xes	0.604	0.711	NA	0.397	0.397	0.071	0.397	0.397	0.232	0.167	0.167
permutations\perml3a10.xes	0.478	0.711	0.478	0.550	0.550	0.003	0.003	0.550	0.550	0.050	0.050
permutations\perml3a3.xes	0.440	0.647	0.440	0.800	1.000	0.071	0.125	0.800	0.321	0.167	0.167
permutations\perml3a5.xes	0.459	0.680	0.459	0.455	0.455	0.020	0.025	0.455	0.455	0.100	0.100
permutations\perml5a10.xes	NA	0.776	NA	NA	NA	0.001	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	0.878	0.878	0.500	0.878	0.878	0.878	0.500	0.621
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	0.500	1.000	1.000	1.000	0.500	1.000
random\randpms1000d2.xes	1.000	0.826	1.000	0.375	0.375	0.500	0.375	0.375	0.937	0.500	0.375
random\randpms1000d3.xes	1.000	0.909	1.000	0.375	0.375	0.500	0.375	0.375	0.913	0.500	0.375
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	0.500	1.000	1.000	1.000	0.500	1.000
random\randpms100d2.xes	1.000	0.907	1.000	0.831	0.831	0.500	0.831	0.831	0.831	0.500	0.591
random\randpms100d3.xes	1.000	0.765	1.000	0.750	0.750	0.500	0.750	0.750	0.973	0.500	0.750
random\rands10000l20m8a10.xes	NA	NA	NA	0.423	0.423	0.001	0.009	0.423	0.230	0.003	0.050
random\rands1000l10m4a5.xes	0.368	0.436	0.368	0.421	0.421	0.025	0.083	0.421	0.271	0.029	0.100
random\rands100l5m2a3.xes	0.471	0.631	0.471	0.405	0.405	0.071	0.167	0.405	0.417	0.100	0.167
reallife\realdocman.xes	NA	NA	NA	0.164	0.371	NA	0.142	0.164	0.087	NA	0.008
reallife\realhospital.xes	NA	NA	NA	0.173	0.254	0.004	0.029	0.173	0.371	NA	0.017
reallife\realincman.xes	0.717	0.973	0.647	0.042	0.045	0.016	0.042	0.042	0.310	NA	0.036
reallife\realoutsourcing.xes	1.000	1.000	NA	NA	NA	0.100	0.084	NA	NA	0.167	0.084
synthetic\a10skip.xes	1.000	1.000	1.000	0.912	0.912	0.500	0.811	0.912	1.000	0.500	0.636
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	0.500	1.000	1.000	1.000	0.500	0.806
synthetic\a5.xes	0.999	0.999	0.632	0.286	0.286	0.500	0.400	0.286	0.478	0.500	0.286
synthetic\a6nfc.xes	0.852	1.000	0.852	0.697	0.697	0.500	0.484	0.697	0.697	0.500	0.484
synthetic\a7.xes	1.000	1.000	1.000	0.872	0.828	0.500	0.393	0.872	NA	0.500	0.280
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	0.500	1.000	1.000	1.000	0.500	0.862
synthetic\betasimplified.xes	0.920	0.702	0.920	0.500	0.500	0.500	0.500	0.500	0.626	0.500	0.500
synthetic\choice.xes	1.000	1.000	1.000	0.571	0.571	0.500	0.571	0.571	0.867	0.500	0.571
synthetic\driverslicense.xes	0.889	1.000	0.889	0.889	0.889	0.500	0.889	0.889	0.889	0.500	0.889
synthetic\driverslicenseloop.xes	NA	0.929	1.000	0.412	0.412	0.500	0.412	0.412	0.799	0.500	0.571
synthetic\herbstfig3p4.xes	0.994	0.994	0.994	0.994	0.994	0.500	0.950	0.994	0.994	0.500	0.734
synthetic\herbstfig5p19.xes	1.000	1.000	1.000	0.589	0.589	0.250	0.589	0.589	0.750	0.250	0.564
synthetic\herbstfig6p18.xes	NA	0.999	0.801	0.400	0.400	0.500	0.400	0.400	0.400	0.500	0.400
synthetic\herbstfig6p31.xes	0.563	1.000	0.563	0.750	0.563	0.500	0.563	0.750	0.563	0.500	0.563
synthetic\herbstfig6p36.xes	0.917	1.000	0.917	0.917	0.917	0.250	0.917	0.917	0.917	0.250	0.917
synthetic\herbstfig6p38.xes	0.426	0.511	0.426	0.624	0.624	0.500	0.400	0.624	1.000	0.500	0.400
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	0.500	1.000	1.000	1.000	0.500	0.400
synthetic\l2l.xes	1.000	0.999	1.000	1.000	0.999	0.500	0.999	NA	0.999	0.500	0.999
synthetic\l2loptional.xes	1.000	1.000	1.000	0.400	0.400	0.500	0.400	0.400	0.894	0.500	0.400
synthetic\l2lskip.xes	1.000	0.905	1.000	0.846	0.846	0.500	0.846	0.846	0.846	0.500	0.846
synthetic\prAm6.xes	0.536	NA	0.536	0.758	0.701	0.500	0.674	0.758	NA	0.500	0.723
synthetic\prBm6.xes	0.337	0.337	0.337	0.651	0.633	0.500	0.541	0.651	0.563	0.500	0.746
synthetic\prCm6.xes	NA	NA	NA	NA	NA	0.249	0.439	NA	NA	NA	NA
synthetic\prDm6.xes	NA	NA	NA	0.350	0.358	0.500	0.276	0.350	0.326	NA	NA
synthetic\prEm6.xes	NA	NA	0.394	0.380	0.404	0.500	0.603	0.380	0.391	NA	0.773
synthetic\prFm6.xes	NA	NA	0.652	0.169	0.205	0.500	0.177	0.169	0.211	NA	NA
synthetic\prGm6.xes	NA	NA	0.866	0.316	0.387	0.500	0.058	0.316	0.435	NA	NA

Table 5.11: Results of the Fodina conformance checking experiment. This table lists the results for the Behavioral Recall metric.

Algorithm	A	A++	A6	HM5	HM6	FHM6	F	$HM5_L$	$HM6_L$	$FHM6_L$	F_L
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						r_B					
permutations\perml10a3.xes	1.000	1.000	NA	0.996	0.996	0.699	0.996	0.996	0.917	1.000	1.000
permutations\perml3a10.xes	1.000	1.000	1.000	0.800	0.800	0.721	0.732	0.800	0.800	1.000	1.000
permutations\perml3a3.xes	1.000	1.000	1.000	0.600	0.667	0.815	0.689	0.600	0.844	1.000	1.000
permutations\perml3a5.xes	1.000	1.000	1.000	0.800	0.800	0.854	0.848	0.800	0.800	1.000	1.000
permutations\perml5a10.xes	1.000	1.000	1.000	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	1.000	1.000	1.000	1.000	1.000	1.000	0.999	1.000
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	0.714	0.929	0.714	1.000	1.000	1.000	1.000	1.000	0.914	1.000	1.000
random\randpms1000d3.xes	0.725	0.771	0.725	1.000	1.000	1.000	1.000	1.000	0.962	0.996	1.000
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.841	0.884	0.841	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d3.xes	0.808	0.906	0.808	1.000	1.000	1.000	1.000	1.000	0.979	0.994	1.000
random\rands10000l20m8a10.xes	1.000	1.000	1.000	0.954	0.954	NA	0.911	0.954	NA	NA	1.000
random\rands1000l10m4a5.xes	1.000	1.000	1.000	0.914	0.914	0.863	0.910	0.914	0.764	0.849	1.000
random\rands100l5m2a3.xes	1.000	1.000	1.000	0.917	0.917	0.881	0.868	0.917	0.669	0.916	1.000
reallife\realdocman.xes	NA	NA	NA	0.666	0.673	NA	0.967	0.668	0.626	NA	0.991
reallife\realhospital.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
reallife\realincman.xes	0.630	0.349	0.631	0.818	0.794	0.885	0.965	0.809	0.584	NA	0.998
reallife\realoutsourcing.xes	0.348	0.002	0.004	NA	0.008	0.827	0.977	NA	0.126	0.695	0.977
synthetic\a10skip.xes	0.946	0.946	0.946	1.000	1.000	1.000	1.000	1.000	0.946	1.000	1.000
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a5.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.937	1.000	1.000
synthetic\a6nfc.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.990	1.000	1.000	0.990	0.990
synthetic\a7.xes	1.000	1.000	1.000	0.957	1.000	0.981	0.940	0.957	NA	0.985	0.994
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\betasimplified.xes	0.836	0.881	0.836	1.000	1.000	1.000	1.000	1.000	0.961	1.000	1.000
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.924	1.000	1.000
synthetic\driverslicense.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicenseloop.xes	NA	0.991	0.917	1.000	1.000	1.000	1.000	1.000	0.869	1.000	1.000
synthetic\herbstfig3p4.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig5p19.xes	0.743	0.828	0.743	1.000	1.000	1.000	1.000	1.000	0.836	1.000	1.000
synthetic\herbstfig6p18.xes	NA	0.795	0.734	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p31.xes	1.000	1.000	1.000	0.833	1.000	1.000	1.000	0.833	1.000	1.000	1.000
synthetic\herbstfig6p36.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p38.xes	1.000	1.000	1.000	0.875	0.875	1.000	0.963	0.875	0.750	1.000	1.000
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.984	1.000
synthetic\l2l.xes	1.000	1.000	1.000	0.850	1.000	1.000	1.000	0.850	1.000	1.000	1.000
synthetic\l2loptional.xes	1.000	1.000	1.000	0.911	1.000	1.000	1.000	0.911	0.918	1.000	1.000
synthetic\l2lskip.xes	0.872	1.000	0.872	0.883	1.000	1.000	1.000	0.883	1.000	1.000	1.000
synthetic\prAm6.xes	0.869	NA	0.869	0.923	0.936	0.939	0.941	0.923	NA	NA	0.999
synthetic\prBm6.xes	0.982	0.983	0.982	0.968	0.979	0.968	0.956	0.968	0.979	0.954	0.998
synthetic\prCm6.xes	NA	NA	NA	0.614	0.606	0.594	0.685	0.614	0.602	NA	NA
synthetic\prDm6.xes	0.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prEm6.xes	NA	NA	0.373	0.768	0.726	NA	0.765	0.766	0.703	NA	NA
synthetic\prFm6.xes	0.000	NA	NA	NA	0.784	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	0.000	NA	NA	0.733	NA	NA	NA	0.733	NA	NA	NA

Table 5.12: Results of the Fodina conformance checking experiment. This table lists the results for the Weighted Behavioral Precision metric.

Algorithm	A	A++	A6	HM5	HM6	FHM6	F	$HM5_L$	$HM6_L$	$FHM6_L$	F_L
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						p_B^w					
permutations\perml10a3.xes	0.609	0.764	NA	0.893	0.893	0.804	0.893	0.893	0.933	0.815	0.815
permutations\perml3a10.xes	0.211	0.395	0.211	0.407	0.407	0.526	0.577	0.407	0.407	0.429	0.429
permutations\perml3a3.xes	0.393	0.573	0.393	0.783	0.852	0.680	0.861	0.783	0.691	0.648	0.648
permutations\perml3a5.xes	0.313	0.500	0.313	0.559	0.559	0.625	0.619	0.559	0.559	0.556	0.556
permutations\perml5a10.xes	0.290	0.531	0.290	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	0.929	0.929	NA	0.929	0.929	0.929	NA	0.791
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	1.000	0.764	1.000	0.830	0.830	0.830	0.830	0.830	0.608	0.613	0.753
random\randpms1000d3.xes	0.789	0.732	0.789	0.709	0.709	NA	0.709	0.709	0.350	NA	0.542
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.867	0.838	0.867	0.886	0.886	0.886	0.886	0.886	0.886	0.620	0.827
random\randpms100d3.xes	0.984	0.699	0.984	0.896	0.896	0.896	0.896	0.896	0.775	0.733	0.753
random\rands10000l20m8a10.xes	0.149	0.161	0.149	0.170	0.170	NA	NA	0.170	NA	NA	NA
random\rands1000l10m4a5.xes	0.324	0.389	0.324	0.437	0.435	0.411	0.430	0.435	0.413	0.413	0.403
random\rands100l5m2a3.xes	0.439	0.597	0.439	0.712	0.712	0.674	0.696	0.712	0.681	0.663	0.656
reallife\realdocman.xes	NA	NA	NA	NA	NA	NA	0.391	NA	NA	NA	NA
reallife\realhospital.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
reallife\realincman.xes	0.232	0.559	0.161	0.382	0.399	NA	0.681	0.375	0.324	NA	0.479
reallife\realoutsourcing.xes	0.896	0.089	0.086	NA	NA	NA	0.921	NA	NA	NA	NA
synthetic\a10skip.xes	0.946	0.946	0.946	1.000	1.000	1.000	1.000	1.000	0.946	0.789	0.900
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.803	0.853
synthetic\a5.xes	1.000	1.000	0.575	1.000	1.000	1.000	1.000	1.000	0.972	0.797	0.819
synthetic\a6nfc.xes	0.870	1.000	0.870	0.870	0.870	0.870	0.808	0.870	0.870	0.873	0.808
synthetic\a7.xes	1.000	1.000	1.000	0.967	0.804	0.826	0.944	0.967	NA	0.712	0.699
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.846	0.920
synthetic\betasimplified.xes	0.683	0.601	0.683	0.850	0.850	0.850	0.850	0.850	0.805	0.850	0.850
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.801	1.000	1.000
synthetic\driverslicense.xes	0.903	1.000	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903
synthetic\driverslicenseloop.xes	NA	0.921	0.899	0.893	0.893	0.893	0.893	0.893	0.647	0.593	0.691
synthetic\herbstfig3p4.xes	0.992	0.992	0.992	0.992	0.992	0.992	0.983	0.992	0.992	0.842	0.820
synthetic\herbstfig5p19.xes	0.873	0.885	0.873	0.902	0.902	0.902	0.902	0.902	0.886	0.802	0.776
synthetic\herbstfig6p18.xes	NA	0.773	0.466	0.971	0.971	0.971	0.971	0.971	0.971	0.971	0.971
synthetic\herbstfig6p31.xes	0.558	1.000	0.558	0.513	0.558	0.558	0.558	0.513	0.558	0.558	0.558
synthetic\herbstfig6p36.xes	0.976	1.000	0.976	0.976	0.976	0.976	0.976	0.976	0.976	0.976	0.976
synthetic\herbstfig6p38.xes	0.428	0.510	0.428	0.836	0.836	0.626	0.578	0.836	NA	0.719	0.598
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.613	0.741
synthetic\l2l.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	NA	1.000	1.000	1.000
synthetic\l2loptional.xes	1.000	1.000	1.000	0.873	1.000	1.000	1.000	0.873	0.918	1.000	1.000
synthetic\l2lskip.xes	1.000	0.887	1.000	0.829	1.000	1.000	1.000	0.829	1.000	1.000	1.000
synthetic\prAm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prBm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prCm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prDm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prEm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prFm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Table 5.13: Results of the Fodina conformance checking experiment. This table lists the results for the Weighted Behavioral Generalization metric.

Algorithm	α	$\alpha++$	$\alpha6$	HM5	HM6	FHM6	F	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML	PNML
Event Log						g_B^w					
permutations\perml10a3.xes	0.609	0.764	NA	0.893	0.893	0.804	0.893	0.893	0.933	0.815	0.815
permutations\perml3a10.xes	0.211	0.395	0.211	0.407	0.407	0.526	0.577	0.407	0.407	0.429	0.429
permutations\perml3a3.xes	0.393	0.573	0.393	0.783	0.852	0.680	0.861	0.783	0.691	0.648	0.648
permutations\perml3a5.xes	0.313	0.500	0.313	0.559	0.559	0.625	0.619	0.559	0.559	0.556	0.556
permutations\perml5a10.xes	0.290	0.531	0.290	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a3.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
permutations\perml5a5.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d1.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d2.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
random\randpms10000d3.xes	NA	NA	NA	0.929	0.929	NA	0.929	0.929	0.929	NA	0.791
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	1.000	0.764	1.000	0.830	0.830	0.830	0.830	0.830	0.608	0.613	0.753
random\randpms1000d3.xes	0.789	0.732	0.789	0.709	0.709	NA	0.709	0.709	0.350	NA	0.542
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.867	0.838	0.867	0.886	0.886	0.886	0.886	0.886	0.886	0.620	0.827
random\randpms100d3.xes	0.984	0.699	0.984	0.896	0.896	0.896	0.896	0.896	0.775	0.733	0.753
random\rands10000l20m8a10.xes	0.149	0.161	0.149	0.170	0.170	NA	NA	0.170	NA	NA	NA
random\rands1000l10m4a5.xes	0.324	0.389	0.324	0.437	0.435	0.411	0.430	0.435	0.413	0.413	0.403
random\rands100l5m2a3.xes	0.439	0.597	0.439	0.712	0.712	0.674	0.696	0.712	0.681	0.663	0.656
reallife\realdocman.xes	NA	NA	NA	NA	NA	NA	0.391	NA	NA	NA	NA
reallife\realhospital.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
reallife\realincman.xes	0.232	0.559	0.161	0.382	0.399	NA	0.681	0.375	0.324	NA	0.479
reallife\realoutsourcing.xes	0.896	0.089	0.086	NA	NA	NA	0.921	NA	NA	NA	NA
synthetic\a10skip.xes	0.946	0.946	0.946	1.000	1.000	1.000	1.000	1.000	0.946	0.789	0.900
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.803	0.853
synthetic\a5.xes	1.000	1.000	0.575	1.000	1.000	1.000	1.000	1.000	0.972	0.797	0.819
synthetic\a6nfc.xes	0.870	1.000	0.870	0.870	0.870	0.870	0.808	0.870	0.870	0.873	0.808
synthetic\a7.xes	1.000	1.000	1.000	0.967	0.804	0.826	0.944	0.967	NA	0.712	0.699
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.846	0.920
synthetic\betasimplified.xes	0.683	0.601	0.683	0.850	0.850	0.850	0.850	0.850	0.805	0.850	0.850
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.801	1.000	1.000
synthetic\driverslicense.xes	0.903	1.000	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903	0.903
synthetic\driverslicenseloop.xes	NA	0.921	0.899	0.893	0.893	0.893	0.893	0.893	0.647	0.593	0.691
synthetic\herbstfig3p4.xes	0.992	0.992	0.992	0.992	0.992	0.992	0.983	0.992	0.992	0.842	0.820
synthetic\herbstfig5p19.xes	0.873	0.885	0.873	0.902	0.902	0.902	0.902	0.902	0.886	0.802	0.776
synthetic\herbstfig6p18.xes	NA	0.773	0.466	0.971	0.971	0.971	0.971	0.971	0.971	0.971	0.971
synthetic\herbstfig6p31.xes	0.558	1.000	0.558	0.513	0.558	0.558	0.558	0.513	0.558	0.558	0.558
synthetic\herbstfig6p36.xes	0.976	1.000	0.976	0.976	0.976	0.976	0.976	0.976	0.976	0.976	0.976
synthetic\herbstfig6p38.xes	0.428	0.510	0.428	0.836	0.836	0.626	0.578	0.836	NA	0.719	0.598
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000	0.613	0.741
synthetic\l2l.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	NA	1.000	1.000	1.000
synthetic\l2loptional.xes	1.000	1.000	1.000	0.873	1.000	1.000	1.000	0.873	0.918	1.000	1.000
synthetic\l2lskip.xes	1.000	0.887	1.000	0.829	1.000	1.000	1.000	0.829	1.000	1.000	1.000
synthetic\prAm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prBm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prCm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prDm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prEm6.xes	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prFm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
synthetic\prGm6.xes	1.000	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA

Table 5.14: Results of the Fodina conformance checking experiment. This table lists the results for the ICS Fitness metric.

Algorithm	HM5	HM6	FHM6	F	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	HNET	HNET	CNET	CNET	HNET	HNET	CNET	CNET
Event Log	ICS							
permutations\perml10a3.xes	0.996	0.996	0.959	0.996	0.996	0.894	1.000	1.000
permutations\perml3a10.xes	-2.116	-2.116	0.969	0.998	-2.116	-2.116	1.000	1.000
permutations\perml3a3.xes	-0.200	0.565	0.944	0.527	-0.200	0.772	1.000	1.000
permutations\perml3a5.xes	-0.224	-0.224	0.952	0.990	-0.224	-0.224	1.000	1.000
permutations\perml5a10.xes	-0.830	-0.830	0.960	0.929	-0.830	-0.830	1.000	1.000
permutations\perml5a3.xes	0.945	0.945	0.943	0.945	0.945	0.881	1.000	1.000
permutations\perml5a5.xes	0.859	0.859	0.944	0.986	0.859	0.854	1.000	1.000
random\randpms10000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.875	1.000
random\randpms10000d2.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms10000d3.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	1.000	1.000	1.000	1.000	1.000	0.741	0.929	1.000
random\randpms1000d3.xes	1.000	1.000	1.000	1.000	1.000	0.960	0.845	1.000
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.998	1.000
random\randpms100d3.xes	1.000	1.000	1.000	1.000	1.000	0.979	0.929	1.000
random\rands10000l20m8a10.xes	0.964	0.964	0.999	0.992	0.964	-2.497	0.570	1.000
random\rands1000l10m4a5.xes	0.952	0.952	0.994	0.989	0.952	-0.392	0.744	1.000
random\rands100l5m2a3.xes	0.913	0.913	0.958	0.970	0.913	-0.233	0.994	1.000
reallife\realdocman.xes	0.662	0.669	0.986	0.967	0.662	0.399	0.998	0.996
reallife\realhospital.xes	0.768	0.606	0.979	0.969	0.768	0.480	0.999	0.999
reallife\realincman.xes	0.823	0.795	0.959	0.965	0.823	0.552	1.000	0.998
reallife\realoutsourcing.xes	NA	-2.844	0.500	0.999	NA	-3.781	0.513	0.999
synthetic\a10skip.xes	1.000	1.000	1.000	1.000	1.000	0.945	1.000	1.000
synthetic\a12.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a5.xes	1.000	1.000	1.000	1.000	1.000	0.937	0.999	1.000
synthetic\a6nfc.xes	1.000	1.000	1.000	0.990	1.000	1.000	0.998	0.990
synthetic\a7.xes	0.957	0.999	0.999	0.950	0.957	NA	0.998	0.994
synthetic\a8.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\betasimplified.xes	1.000	1.000	1.000	1.000	1.000	0.961	1.000	1.000
synthetic\choice.xes	1.000	1.000	1.000	1.000	1.000	0.923	1.000	1.000
synthetic\driverslicense.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicenseloop.xes	1.000	1.000	1.000	1.000	1.000	0.866	1.000	1.000
synthetic\herbstfig3p4.xes	1.000	1.000	1.000	0.998	1.000	1.000	0.942	1.000
synthetic\herbstfig5p19.xes	1.000	1.000	1.000	1.000	1.000	0.671	0.999	1.000
synthetic\herbstfig6p18.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p31.xes	0.500	1.000	1.000	1.000	0.500	1.000	1.000	1.000
synthetic\herbstfig6p36.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p38.xes	0.552	0.552	0.875	0.963	0.552	-0.066	0.999	1.000
synthetic\herbstfig6p41.xes	1.000	1.000	1.000	1.000	1.000	1.000	0.999	1.000
synthetic\l2l.xes	0.848	1.000	1.000	1.000	0.848	1.000	1.000	1.000
synthetic\l2loptional.xes	0.911	1.000	1.000	1.000	0.911	0.917	1.000	1.000
synthetic\l2lskip.xes	0.882	1.000	1.000	1.000	0.882	1.000	1.000	1.000
synthetic\prAm6.xes	0.924	0.936	0.972	0.952	0.924	NA	0.990	0.999
synthetic\prBm6.xes	0.966	0.979	0.991	0.970	0.966	0.979	0.992	0.998
synthetic\prCm6.xes	-0.245	-0.291	0.053	0.058	-0.245	-0.338	0.669	0.981
synthetic\prDm6.xes	-0.187	-0.424	0.060	-0.124	-0.187	-0.487	0.527	1.000
synthetic\prEm6.xes	0.220	0.073	0.464	0.394	0.220	0.006	0.569	0.999
synthetic\prFm6.xes	0.324	0.326	0.509	0.467	0.324	0.315	0.450	1.000
synthetic\prGm6.xes	0.086	-0.155	0.371	0.308	0.086	-0.241	0.571	0.999

Table 5.15: Results of the Fodina conformance checking experiment. This table lists the results for the simplicity metrics.

Algorithm	A	A + +	A6	HM5	HM6	FHM6	F	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml	pnml
Event Log						#a / #p / #t					
permutations\perml10a3.xes	2 / 2 / 5	10 / 3 / 5	NA / NA / NA	26 / 11 / 11	26 / 11 / 11	72 / 19 / 31	26 / 11 / 11	26 / 11 / 11	62 / 19 / 17	70 / 25 / 35	40 / 10 / 20
permutations\perml3a10.xes	2 / 2 / 12	24 / 3 / 12	2 / 2 / 12	82 / 32 / 32	82 / 32 / 32	1374 / 54 / 402	1294 / 34 / 362	82 / 32 / 32	82 / 32 / 32	504 / 144 / 252	264 / 24 / 132
permutations\perml3a3.xes	2 / 2 / 5	10 / 3 / 5	2 / 2 / 5	10 / 4 / 5	14 / 8 / 5	72 / 19 / 31	36 / 10 / 13	10 / 4 / 5	39 / 14 / 13	70 / 25 / 35	40 / 10 / 20
permutations\perml3a5.xes	2 / 2 / 7	14 / 3 / 7	2 / 2 / 7	42 / 17 / 17	42 / 17 / 17	214 / 29 / 77	174 / 19 / 57	42 / 17 / 17	42 / 17 / 17	154 / 49 / 77	84 / 14 / 42
permutations\perml5a10.xes	2 / 2 / 12	24 / 3 / 12	2 / 2 / 12	NA / NA / 32	NA / NA / 32	NA / NA / 1326	NA / NA / 452	NA / NA / 32	NA / NA / 32	NA / NA / 252	NA / NA / 132
permutations\perml5a3.xes	NA / NA / 5	NA / NA / 5	NA / NA / 5	NA / NA / 11	NA / NA / 11	NA / NA / 31	NA / NA / 11	NA / NA / 11	NA / NA / 17	NA / NA / 35	NA / NA / 20
permutations\perml5a5.xes	NA / NA / 7	NA / NA / 7	NA / NA / 7	NA / NA / 17	NA / NA / 17	NA / NA / 89	NA / NA / 49	NA / NA / 17	NA / NA / 33	NA / NA / 77	NA / NA / 42
random\randpms10000d1.xes	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 22	NA / NA / 8	NA / NA / 8	NA / NA / 8	NA / NA / 26	NA / NA / 14
random\randpms10000d2.xes	NA / NA / 0	NA / NA / 16	NA / NA / 16	NA / NA / 18	NA / NA / 18	NA / NA / 56	NA / NA / 18	NA / NA / 18	NA / NA / 18	NA / NA / 56	NA / NA / 18
random\randpms10000d3.xes	NA / NA / 38	NA / NA / 38	NA / NA / 38	96 / 40 / 43	96 / 40 / 43	270 / 127 / 130	96 / 40 / 43	96 / 40 / 43	96 / 40 / 43	346 / 134 / 162	146 / 46 / 73
random\randpms1000d1.xes	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	42 / 21 / 21	14 / 7 / 7	14 / 7 / 7	14 / 7 / 7	42 / 21 / 21	14 / 7 / 7
random\randpms1000d2.xes	74 / 25 / 14	33 / 14 / 14	78 / 26 / 14	54 / 20 / 26	54 / 20 / 26	114 / 50 / 56	54 / 20 / 26	54 / 20 / 26	30 / 15 / 14	122 / 51 / 60	64 / 22 / 32
random\randpms1000d3.xes	228 / 81 / 51	201 / 67 / 51	236 / 83 / 51	188 / 71 / 83	188 / 71 / 83	420 / 187 / 199	188 / 71 / 83	188 / 71 / 83	130 / 60 / 54	580 / 202 / 267	298 / 85 / 149
random\randpms100d1.xes	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / 28 / 29	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / 28 / 29	18 / 8 / 9
random\randpms100d2.xes	46 / 19 / 18	44 / 18 / 18	46 / 19 / 18	46 / 19 / 22	46 / 19 / 22	126 / 59 / 62	46 / 19 / 22	46 / 19 / 22	46 / 19 / 22	134 / 60 / 66	56 / 21 / 28
random\randpms100d3.xes	56 / 24 / 20	61 / 23 / 20	56 / 24 / 20	56 / 24 / 26	56 / 24 / 26	140 / 66 / 68	56 / 24 / 26	56 / 24 / 26	44 / 21 / 20	174 / 69 / 82	76 / 26 / 38
random\rands10000l20m8a10.xes	2 / 2 / 12	24 / 3 / 12	2 / 2 / 12	82 / 32 / 32	82 / 32 / 32	5860 / 54 / 816	1096 / 34 / 144	82 / 32 / 32	712 / 194 / 122	4041 / 99 / 1125	264 / 24 / 132
random\rands1000l10m4a5.xes	2 / 2 / 7	14 / 3 / 7	NA / 2 / 7	42 / 17 / 17	42 / 17 / 17	220 / 29 / 67	96 / 19 / 29	42 / 17 / 17	169 / 49 / 37	318 / 40 / 121	84 / 14 / 42
random\rands100l5m2a3.xes	2 / 2 / 5	10 / 3 / 5	2 / 2 / 5	26 / 11 / 11	26 / 11 / 11	72 / 19 / 31	42 / 13 / 17	26 / 11 / 11	57 / 19 / 17	92 / 23 / 41	40 / 10 / 20
reallife\realdocman.xes	NA / NA / NA	NA / NA / NA	NA / NA / NA	809 / 108 / 205	621 / 62 / 129	NA / NA / NA	424 / 69 / 208	809 / 108 / 205	1940 / 322 / 432	NA / NA / NA	1841 / 144 / 919
reallife\realhospital.xes	NA / NA / NA	NA / NA / NA	NA / NA / NA	3226 / 442 / 1278	3458 / 466 / 1266	22033 / 3093 / 8137	3240 / 504 / 1575	3226 / 442 / 1278	4295 / 701 / 1425	NA / NA / NA	8999 / 868 / 4499
reallife\realincman.xes	78 / 20 / 18	273 / 50 / 18	70 / 20 / 18	229 / 50 / 80	208 / 55 / 68	669 / 102 / 270	181 / 32 / 90	229 / 50 / 80	314 / 82 / 83	NA / NA / NA	310 / 34 / 155
reallife\realoutsourcing.xes	22 / 10 / 7	25 / 8 / 7	55 / 10 / 7	NA / NA / NA	98 / 33 / 28	160 / 43 / 72	64 / 14 / 32	NA / NA / NA	154 / 46 / 40	383 / 40 / 134	78 / 14 / 39
synthetic\a10skip.xes	30 / 14 / 12	30 / 14 / 12	30 / 14 / 12	30 / 13 / 13	30 / 13 / 13	78 / 38 / 38	32 / 15 / 15	30 / 13 / 13	28 / 13 / 12	86 / 39 / 42	38 / 15 / 19
synthetic\a12.xes	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	90 / 44 / 44	30 / 14 / 14	30 / 14 / 14	30 / 14 / 14	103 / 46 / 50	46 / 17 / 23
synthetic\a5.xes	18 / 8 / 7	18 / 8 / 7	NA / 8 / 7	31 / 13 / 13	31 / 13 / 13	51 / 24 / 24	25 / 11 / 11	31 / 13 / 13	25 / 11 / 11	61 / 25 / 29	34 / 12 / 17
synthetic\a6nfc.xes	18 / 9 / 8	19 / 9 / 8	18 / 9 / 8	20 / 10 / 9	20 / 10 / 9	52 / 25 / 25	24 / 10 / 12	20 / 10 / 9	20 / 10 / 9	60 / 26 / 29	24 / 10 / 12
synthetic\a7.xes	24 / 10 / 9	24 / 10 / 9	24 / 10 / 9	24 / 9 / 10	22 / 9 / 9	73 / 30 / 32	27 / 11 / 12	24 / 9 / 10	NA / NA / NA	133 / 36 / 56	54 / 15 / 27
synthetic\a8.xes	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	66 / 32 / 32	22 / 10 / 10	22 / 10 / 10	22 / 10 / 10	74 / 33 / 36	32 / 12 / 16
synthetic\betasimplified.xes	34 / 15 / 13	45 / 18 / 13	34 / 15 / 13	42 / 17 / 21	42 / 17 / 21	94 / 43 / 47	42 / 17 / 21	42 / 17 / 21	38 / 16 / 19	94 / 43 / 47	42 / 17 / 21
synthetic\choice.xes	24 / 9 / 12	24 / 9 / 12	24 / 9 / 12	48 / 18 / 24	48 / 18 / 24	96 / 42 / 48	48 / 18 / 24	48 / 18 / 24	30 / 12 / 15	96 / 42 / 48	48 / 18 / 24
synthetic\driverslicense.xes	18 / 8 / 9	22 / 10 / 9	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / 28 / 29	18 / 8 / 9	18 / 8 / 9	18 / 8 / 9	58 / NA / NA	18 / 8 / 9
synthetic\driverslicenseloop.xes	NA / NA / NA	34 / 13 / 11	31 / 12 / 11	38 / 15 / 17	38 / 15 / 17	86 / 39 / 41	38 / 15 / 17	38 / 15 / 17	26 / 12 / 12	98 / 40 / 47	46 / 15 / 23
synthetic\herbstfig3p4.xes	26 / 12 / 12	26 / 12 / 12	26 / 12 / 12	26 / 12 / 12	26 / 12 / 12	79 / 38 / 38	28 / 13 / 13	26 / 12 / 12	26 / 12 / 12	103 / 42 / 50	50 / 17 / 25
synthetic\herbstfig5p19.xes	24 / 10 / 8	22 / 9 / 8	24 / 10 / 8	26 / 11 / 12	26 / 11 / 12	58 / 27 / 28	26 / 11 / 12	26 / 11 / 12	21 / 10 / 10	66 / 28 / 32	34 / 12 / 17
synthetic\herbstfig6p18.xes	NA / NA / NA	15 / 6 / 7	13 / 6 / 7	20 / 8 / 10	20 / 8 / 10	50 / 23 / 25	20 / 8 / 10	20 / 8 / 10	20 / 8 / 10	50 / 23 / 25	20 / 8 / 10
synthetic\herbstfig6p31.xes	18 / 5 / 9	29 / 8 / 9	18 / 5 / 9	23 / 7 / 9	18 / 5 / 9	70 / 31 / 35	18 / 5 / 9	23 / 7 / 9	18 / 5 / 9	70 / 31 / 35	18 / 5 / 9
synthetic\herbstfig6p36.xes	24 / 11 / 12	28 / 13 / 12	24 / 11 / 12	24 / 11 / 12	24 / 11 / 12	76 / 37 / 38	24 / 11 / 12	24 / 11 / 12	24 / 11 / 12	76 / 37 / 38	24 / 11 / 12
synthetic\herbstfig6p38.xes	10 / 6 / 7	14 / 7 / 7	10 / 6 / 7	26 / 11 / 9	26 / 11 / 9	58 / 23 / 27	28 / 10 / 14	26 / 11 / 9	35 / 14 / 12	64 / 24 / 29	36 / 12 / 18
synthetic\herbstfig6p41.xes	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	102 / 51 / 48	38 / 19 / 16	38 / 19 / 16	38 / 19 / 16	134 / 55 / 64	74 / 26 / 37
synthetic\l2l.xes	12 / 7 / 6	12 / 6 / 6	14 / 6 / 6	16 / 8 / 6	12 / 6 / 6	36 / 18 / 18	12 / 6 / 6	16 / 8 / 6	12 / 6 / 6	36 / 18 / 18	12 / 6 / 6
synthetic\l2loptional.xes	12 / 6 / 6	12 / 6 / 6	12 / 6 / 6	22 / 10 / 10	20 / 9 / 10	40 / 19 / 20	20 / 9 / 10	22 / 10 / 10	14 / 7 / 7	40 / 19 / 20	20 / 9 / 10
synthetic\l2lskip.xes	14 / 7 / 6	12 / 6 / 6	14 / 7 / 6	16 / 8 / 7	14 / 7 / 7	36 / 18 / 18	14 / 7 / 7	16 / 8 / 7	14 / 7 / 7	36 / 18 / 18	14 / 7 / 7
synthetic\prAm6.xes	3920 / 986 / 363	NA / NA / NA	3937 / 990 / 363	944 / 336 / 416	951 / 369 / 399	4179 / 1404 / 1842	1222 / 429 / 526	944 / 336 / 416	NA / NA / NA	11579 / 2336 / 4818	4470 / 553 / 2235
synthetic\prBm6.xes	951 / 396 / 317	1026 / 402 / 317	951 / 396 / 317	794 / 309 / 336	831 / 358 / 331	2927 / 1122 / 1291	1030 / 393 / 417	794 / 309 / 336	844 / 355 / 339	6408 / 1450 / 2615	2394 / 470 / 1197
synthetic\prCm6.xes	NA / NA / NA	NA / NA / NA	NA / NA / NA	740 / 238 / 360	781 / 275 / 350	5368 / 1328 / 2260	1210 / 408 / 542	740 / 238 / 360	1270 / 390 / 530	NA / NA / NA	NA / NA / NA
synthetic\prDm6.xes	0 / 0 / 0	NA / NA / NA	NA / NA / NA	1290 / 450 / 556	1364 / 552 / 516	13165 / 1950 / 4719	2697 / 730 / 1117	1290 / 450 / 556	1921 / 710 / 686	NA / NA / NA	NA / NA / NA
synthetic\prEm6.xes	NA / NA / NA	NA / NA / NA	3949 / 1137 / 275	761 / 293 / 310	747 / 322 / 288	4501 / 1072 / 1770	1456 / 464 / 603	761 / 293 / 310	896 / 370 / 327	NA / NA / NA	13044 / 489 / 6522
synthetic\prFm6.xes	0 / 0 / 0	NA / NA / NA	8018 / 2510 / 299	NA / 367 / 348	899 / 406 / 330	5761 / 1222 / 2204	2541 / 713 / 1020	892 / 367 / 348	1096 / 476 / 379	NA / NA / NA	NA / NA / NA
synthetic\prGm6.xes	0 / 0 / 0	NA / NA / NA	40021 / 7589 / 335	1046 / 359 / 440	1061 / 423 / 398	7620 / 1473 / 2891	2205 / 571 / 908	1046 / 359 / 440	1416 / 529 / 516	NA / NA / NA	NA / NA / NA

Table 5.16: Results of the Fodina conformance checking experiment. This table lists the results for the Fitting Single Trace Measure metric.

Algorithm	A	A++	A6	HM5$_L$	HM6$_L$	FHM6$_L$	F$_L$
Model	pnml	pnml	pnml	hnet	hnet	flex	cnet
Event Log				PM[1]			
permutations\perml10a3.xes	0.625	0.489	0.940	0.360	0.180	0.430	1.000
permutations\perml3a10.xes	0.981	0.977	1.000	0.820	1.000	1.000	1.000
permutations\perml3a3.xes	0.926	0.852	1.000	0.560	1.000	1.000	1.000
permutations\perml3a5.xes	0.928	0.928	1.000	0.670	1.000	1.000	1.000
permutations\perml5a10.xes	0.902	0.909	0.985	0.690	0.970	1.000	1.000
permutations\perml5a3.xes	0.675	0.691	0.940	0.530	0.760	0.900	1.000
permutations\perml5a5.xes	0.782	0.784	0.959	0.540	0.890	0.970	1.000
random\randpms10000d1.xes	0.442	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms10000d2.xes	0.477	0.305	0.965	0.900	0.910	1.000	1.000
random\randpms10000d3.xes	0.705	0.533	0.983	0.840	0.910	0.980	1.000
random\randpms1000d1.xes	0.846	0.669	1.000	1.000	1.000	1.000	1.000
random\randpms1000d2.xes	0.849	0.763	1.000	1.000	1.000	1.000	1.000
random\randpms1000d3.xes	0.612	0.514	0.974	0.640	0.810	0.840	1.000
random\randpms100d1.xes	1.000	1.000	1.000	1.000	1.000	1.000	1.000
random\randpms100d2.xes	0.450	0.490	0.948	0.840	0.900	0.980	1.000
random\randpms100d3.xes	0.930	0.810	1.000	1.000	1.000	1.000	1.000
random\rands10000l20m8a10.xes	0.055	0.029	0.781	0.040	0.240	0.710	1.000
random\rands1000l10m4a5.xes	0.347	0.267	0.885	0.230	0.430	0.750	1.000
random\rands100l5m2a3.xes	0.770	0.780	0.966	0.600	0.770	0.890	1.000
reallife\realdocman.xes	0.846	0.976	0.999	1.000	1.000	1.000	1.000
reallife\realhospital.xes	0.273	0.360	0.893	0.340	0.550	0.740	1.000
reallife\realincman.xes	0.800	0.951	0.995	1.000	1.000	1.000	1.000
reallife\realoutsourcing.xes	0.631	0.924	0.970	0.920	0.930	0.930	1.000
synthetic\a10skip.xes	0.467	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a12.xes	0.523	0.753	1.000	1.000	1.000	1.000	1.000
synthetic\a5.xes	0.657	0.883	1.000	0.900	1.000	1.000	1.000
synthetic\a6nfc.xes	0.800	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\a7.xes	0.927	0.903	1.000	1.000	1.000	1.000	1.000
synthetic\a8.xes	0.813	0.643	1.000	1.000	1.000	1.000	1.000
synthetic\betasimplified.xes	1.000	0.493	1.000	1.000	1.000	1.000	1.000
synthetic\choice.xes	0.950	0.873	1.000	0.920	1.000	1.000	1.000
synthetic\driverslicense.xes	0.500	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\driverslicenseloop.xes	0.483	0.654	0.953	0.670	0.810	0.780	1.000
synthetic\herbstfig3p4.xes	0.906	0.781	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig5p19.xes	0.883	0.767	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p18.xes	0.283	0.517	0.878	0.750	1.000	0.860	1.000
synthetic\herbstfig6p31.xes	0.293	0.770	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p36.xes	0.450	1.000	1.000	1.000	1.000	1.000	1.000
synthetic\herbstfig6p38.xes	0.850	0.857	1.000	0.560	1.000	1.000	1.000
synthetic\herbstfig6p41.xes	0.750	0.750	1.000	1.000	1.000	1.000	1.000
synthetic\l2l.xes	1.000	1.000	1.000	0.750	1.000	1.000	1.000
synthetic\l2loptional.xes	0.237	0.997	0.972	1.000	1.000	1.000	1.000
synthetic\l2lskip.xes	0.490	1.000	0.944	1.000	1.000	1.000	1.000
synthetic\prAm6.xes	0.959	0.965	1.000	1.000	1.000	1.000	1.000
synthetic\prBm6.xes	0.941	0.959	1.000	1.000	1.000	1.000	1.000
synthetic\prCm6.xes	0.954	0.976	1.000	1.000	1.000	1.000	1.000
synthetic\prDm6.xes	0.968	0.981	1.000	1.000	1.000	1.000	1.000
synthetic\prEm6.xes	0.962	0.971	1.000	0.990	1.000	1.000	1.000
synthetic\prFm6.xes	0.963	0.976	1.000	1.000	1.000	1.000	1.000
synthetic\prGm6.xes	0.968	0.970	1.000	1.000	1.000	1.000	1.000

Chapter 6

Event-Granular Real-Time Decomposed Conformance Analysis

"Streaming in the wind
The smoke from Fuji
Vanishes in the sky;
I know not where
These thoughts of mine go, either."
– The Monk Saigyô

6.1 Introduction

In this chapter, we look at the application of conformance checking techniques in a real-time manner. Conformance checking techniques face some hard and complex challenges in the context of today's organizations. First, the increasing amount of information systems being implemented and applied to provide operational support, drive decisions and assist managers have led to a barrage of data, which should be parsed and analyzed in a manner which is both correct and scalable by conformance checking techniques. Second, given the current turbulent economic environment, stakeholders desire more than ever the timely delivery of reports and warnings, so that conformance checking techniques should no longer be applied in a post-hoc manner, after the actual occurrence of the activities being executed. Third, such techniques should be able to quickly and correctly localize and pinpoint deviating behavior and its root causes. As process models can become very complex, one wishes to highlight misbehaving parts in a running model, together with the ability to “zoom in and out” on these elements. Many conformance checking techniques have mainly been aiming to derive a global quality “metric”, denoting the global fitness or appropriateness of a process model, but without any real attention being applied towards localizing the main points of failure in an understandable manner.

In this chapter, we build upon the work developed in Chapter 3 to propose a novel methodology to support real-time conformance analysis of event-based data streams, which aims to provide an answer to the challenges listed above. Our approach contributes to the current body of work in the following ways. First, we apply state of the art process model decomposition techniques [180] to split a large process model in a series of sub processes in order to gain a significant *speed-up* when verifying events. Second, by applying decomposition techniques, *localizing deviations* and volatile parts of the process models becomes more straightforward, allowing end-users to quickly gain an insight in which parts of the current model are failing or being violated. Third, by combining model decomposition techniques with a fast, event-granular replay technique, we are able to perform the conformance analysis task in a *real-time* manner, thus allowing for the monitoring of incoming events as they are being executed. This is a strong contribution compared to earlier approaches [102–104, 108–112], where conformance checking techniques assume that a full recorded event log is available and where the actual analysis can be time-consuming.

The possible areas of application for our developed approach are manifold. In light of recent financial crises, the importance of the ability to immediately react to external shocks and unforeseen events has become more apparent than ever. Real-time monitoring, fraud detection and governance, risk and compliance (GRC) verification, and trading system failure protection all provide suitable contexts to apply our proposed technique. We apply our technique on a case example of a large bank transfer process to illustrate the validity of our contribution.

6.2 Preliminaries

This section provides an overview of preliminary definitions and concepts, as well as an overview of related work.

6.2.1 Related Work

As we have discussed already before (see the introductory chapter), conformance checking techniques are devoted to quantify the quality of a process model in describing an event log. In [99], a "fitness" metric is presented to describe the extent to which event traces can be associated with valid execution paths in the process model, and an "appropriateness" metric is proposed to assess whether the process model describes the observed behavior accurately enough. The aforementioned approach *replays* the traces of the log in the model to evaluate these metrics. One of the drawbacks of this approach is that for *undeterministic* models, the heuristics used in the replay may lead to overestimating the metrics, due to the artificial creation of superfluous tokens in the model. Several solutions have been proposed to overcome this issue. Weidlich et al. propose a system to check process model consistency based on "behavioral profiles" [103]—which can be derived in a straightforward and efficient manner but with loss of some granularity regarding the exact traces which can be accepted by the model at hand. Adriansyah et al. propose an alternative approach where the concept of "alignments" are introduced in order to match an event trace with a path through the model as closely as possible [112].

In [180], various *decomposition* approaches to improve process discovery and conformance checking tasks have been proposed. In [181], the notion of passages is used to decompose a process model and/or event log into smaller parts to speed-up process discovery and conformance checking. This approach has been generalized in [180] where it is shown that any event-granular process discovery and conformance checking tasks can be decomposed as long as the different process fragments (i.e. the submodels) only share uniquely-labeled activities. We apply this approach in this chapter, but utilize a Refined Process Structure Tree (RPST) based decomposition method as outlined in [158, 182], as the hierarchical topological structure provided by this decomposition allows to enable additional analytical tasks not considered before, such as zooming in and out on various parts of the process model being monitored.

Our methodology also bears similarities with the fields of Complex Event Processing (CEP) [148, 149] and Business Activity Monitoring (BAM) [183, 184]. Although these methodologies also support the (near-)real-time monitoring of events, our approach differentiates itself in two ways. First, our approach stays in the general realm of process mining by starting from a process model and comparing this against a stream of incoming events which can be related to several running process instances. Second, as we apply a decomposition strategy over the given process model, this allows to immediately relate violations or discrepancies to specific areas within this model, thus improving the localization of the root-causes behind such deviations.

Our conformance analysis methodology is applicable on all process models on which event-granular semantics can be defined and which can be meaningfully decomposed into a series of submodels. We continue to apply Petri nets throughout this chapter as the representational language for prescriptive process models.

6.2.2 Definitions

The concepts of an event log, Petri net and Workflow net are applied in this chapter, similar as were defined in the introductory chapter. We also define the following additional preliminary definitions.

Definition 6.1. Workflow graph, System net, Full firing sequence. Given a Petri net $PN = (P, T, F)$, the workflow graph is defined as the structural directed graph $G = (V, E)$ with $V = P \cup T$ and $E = F$. A system net is a triplet defined over a given Petri net $SN = (PN, m_i, m_o)$ where m_i and m_o define the initial and final markings of the Petri net, respectively. $(PN, m_1)[\sigma\rangle(PN, m_2)$ denotes that a sequence of transitions $\sigma \in T$ is enabled and can be fired starting from marking m_1, resulting in marking m_2. □

Definition 6.2. Event stream. For the purpose of our real-time conformance analysis methodology, we define an event stream $ES = \langle e_1, e_2, \ldots \rangle$ as a sequence (finite or infinite) of arriving events. Events $e \in ES$ are expressed as a tuple $e = (id, act, time)$ with $act : ES \rightarrow A_L$ a function denoting the activity for an event, function $time : ES \rightarrow \mathbb{R}$ denoting the timestamp and function $id : ES \rightarrow \mathbb{N}$ denoting the case identifier. It is trivial to convert an event log L to an event stream ES if a global order relation can be established over the recorded activities in the event log (i.e. based on a timestamp) and vice versa. □

6.3 Methodology

This section presents the developed real-time decomposed conformance analysis approach. Figure 6.1 provides a schematic overview of the approach, which can be split up in four phases, explained in the next subsections.

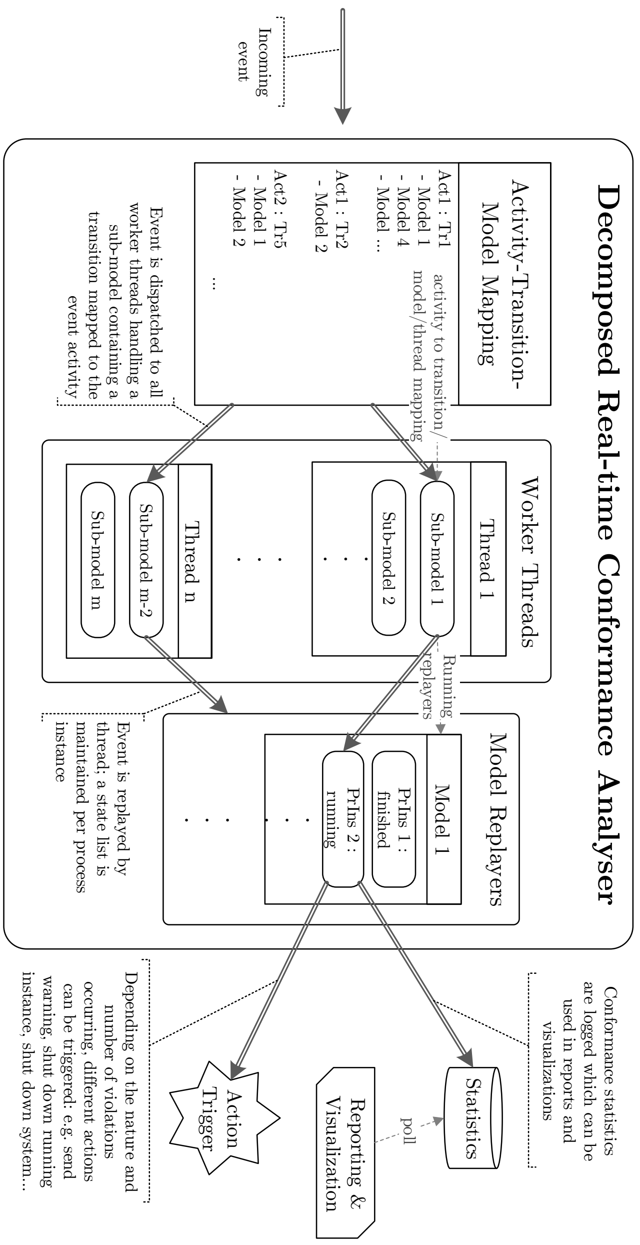

Incoming event
Decomposed Real-time Conformance Analyser
Activity-Transition-Model Mapping
Act1 : Tr1
- Model 1
- Model 4
- Model ...
Act1 : Tr2
- Model 2
Act2 : Tr5
- Model 1
- Model 2
...
activity to transition/ model/thread mapping
Event is dispatched to all worker threads handling a sub-model containing a transition mapped to the event activity
Worker Threads
Thread 1
Sub-model 1
Sub-model 2
Thread n
Sub-model m-2
Sub-model m
Running replayers
Model Replayers
Model 1
PrIns 1 : finished
PrIns 2 : running
Event is replayed by thread; a state list is maintained per process instance
Conformance statistics are logged which can be used in reports and visualizations
Statistics
poll
Reporting & Visualization
Action Trigger
Depending on the nature and number of violations occurring, different actions can be triggered: e.g. send warning, shut down running instance, shut down system...

Figure 6.1: Architectural overview of the developed real-time decomposed conformance analysis technique.

6.3.1 Phase 1: Decomposition

The first phase of the proposed methodology entails *decomposition*. Formally, the overall system net $SN = (PN, m_i, m_o)$ is broken down into a collection of subnets $\{SN^1, SN^2, \ldots SN^n\}$ such that the union of these subnets yields the original system net $SN = \bigcup_{1 \leqslant i \leqslant n} SN^i$. By means of decomposing the original model into a set of subnets we aim to achieve the following goals. First, fragment the conformance problems into a set of more comprehensive semantic elements aiding on the diagnosis. Second, restrict the possible pernicious effects of the heuristics decisions taken during the conformance analysis (see Phase 3 below). Third, speed-up the analysis compared with non-decomposed conformance checking techniques.

Due to the final goal of analyzing conformance, not all possible decomposition approaches are appropriate for this task. Only those *valid* decompositions that preserve the conformance integrity should be considered [180]. That is, given the original net and the decomposed version, *the original net perfectly conforms iff all the subnets in the decomposed setting perfectly conforms.* In other words, no conformance anomalies should be lost or introduced in the transition from the overall model to the decomposed one. In [180], the authors define a valid decomposition—applicable on Petri nets—as the decomposition that satisfies the following conditions:

1. Each arc of the overall net belongs to exactly one of the submodels, i.e.,$F = \bigcup_{1 \leqslant i \leqslant n} F^i$where $F^i \cap F^j = \emptyset$ for $1 \leqslant i < j \leqslant n$.
2. Each place of the overall net belongs to exactly one of the submodels, i.e.,$P = \bigcup_{1 \leqslant i \leqslant n} P^i$where $P^i \cap P^j = \emptyset$ for $1 \leqslant i < j \leqslant n$.
3. Silent transitions appears in precisely one of the subnets, i.e., $\forall t \in T \setminus T_v(SN) : [|\{1 \leqslant i \leqslant n \mid t \in T^i\}| = 1]$, where $T_v(SN)$ stands for the set of visible transitions (i.e., non-silent) of SN, i.e. $T_v(SN) = \{t \in T | \mu(t) \neq s\}$.
4. Non-silent, duplicate transitions appear in precisely one of the subnets, i.e., $\forall t \in T_v(SN) \setminus T_v^u(SN) : [|\{1 \leqslant i \leqslant n \mid t \in T^i\}| = 1]$, where $T_v^u(SN)$ stands for the set of visible transitions with unique label (i.e., non-silent and non-duplicate) of SN, i.e. $T_v^u(SN) = \{t \in T | \mu(t) \neq s \wedge \nexists x \in T : x \neq t \wedge \mu(x) = \mu(t)\}$.
5. Non-silent, non-duplicate transitions may appear in multiple subnets, i.e., $\forall t \in T_v^u(SN) : [|\{1 \leqslant i \leqslant n \mid t \in T^i\}| \geqslant 1]$.

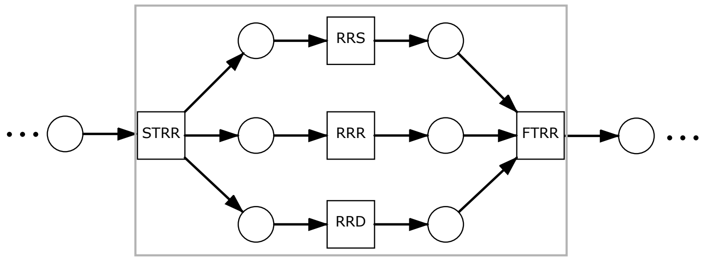


Figure 6.2: *"Open and register transaction"* SESE-component from the case example in Figure 6.5. *STRR* and *FTRR* are the *entry* and *exit boundary* nodes of the SESE-component, respectively. The rest of places and transitions are *interior* nodes of the SESE-component.

In other words, all the elements in the original Petri net model must belong to a submodel, but only unique visible transitions can be shared among several submodels. In [180], the authors prove that any valid decomposition satisfying the aforementioned conditions captures all the conformance problems of the overall model, and not more, i.e., it preserves the conformance integrity.

In [158], an approach based on SESE-components (Single-Entry Single-Exit) is presented, i.e. subgraphs in the workflow graph defined over a system net having single entry and exit boundary nodes [185]. The decomposition of the workflow graph of a Petri net into SESE-components is well-studied and provides a valid means to perform process model decomposition. In addition, SESE-components represent a well-defined and understandable part of the process model, with the added benefit that it is possible to define a hierarchical structure among the model fragments which allows to navigate through the different levels of granularity, so that a SESE-component perfectly reflects the idea of subprocesses within the main process. Figure 6.2 depicts an example SESE-component for the illustrative case shown in Figure 6.4, obtained using the technique proposed in [158]. Note that this techniques can be combined with a user-supervised post-processing step in order to obtain components that better fulfill the domain-aware monitoring.

6.3.2 Phase 2: Event Dispatching

Once a system net has been decomposed into a set of submodels, this collection of models is passed to a central *event dispatcher*, which also serves to listen for incoming events. For each submodels, it is examined whether it contains a transition t which maps to the incoming event e, i.e. $\exists t \in T : \mu(t) = act(e)$. If it does, this indicates that the event at hand should be replayed on this particular submodel (multiple such submodels can be found), and the event is passed forward to this model fragment.

It is possible to decouple the “worker”-instances for each model fragment in a distributed fashion, with each model fragment running on separate machines. For the purpose of our prototype, we have implemented a multi-threaded architecture where a number of worker threads smaller than or equal to the number of process fragments is spawned, with each worker thread overseeing the handling of one or more process fragments. This approach allows for the concurrent handling of event checking over the different model fragments.

6.3.3 Phase 3: Replay

Once it is determined which process model fragment(s) should parse the incoming event, the actual *replay* of this event on each such fragment is performed. For each process model fragment, a state list is maintained denoting the current marking reached by the currently-running process instances. When an event e is queued for replay by a process fragment, the state linked to process instance $\mathtt{id}(e)$ is progressed by investigating whether there exists an enabled transition $\exists t \in T : [\mu(t) = \mathtt{act}(e) \wedge \mathtt{enabled}(t)]$. The outcome of this evaluation determines if the process model is showing discrepancies or not.

Some additional remarks should be provided at this point. First of all, we note that we apply a heuristic, event-granular replayer similar to the one developed in Chapter 3. The reasoning behind the choice to opt for a replayer playing the token game instead of an alternative approach such as alignment or behavioral profile based techniques [103, 112] are twofold:

1. First, alignment and behavioral profile based replayers perform their analysis on a trace, rather than event level, meaning that a complete process instance needs to finalize in order to align the log trace with a process model transition sequence As we are dealing with event streams which need to be analyzed in a real-time manner, an event-granular replay strategy is required. Additionally, the behavioral profile approach does not specify how duplicate tasks can be dealt with.

2. Second, alternative approaches suffer from scalability issues which make them unsuitable in a real-time context. Subsequently, the replay procedure applied here is formalized in Algorithm 6.1.

A second remark entails the way decision points are resolved by the replayer. Put briefly, whenever multiple (enabled) transitions are mapped to the same event log activity within a process model and/or whenever multiple invisible activities are enabled, the replayer needs to determine which transition to execute to handle the activity at hand. Note that—in extreme edge cases—it is possible that the forced firing of a non-enabled transition should be preferred if this avoids several other violations later in the event trace, as we have seen in Chapter 3. In Algorithm 6.1, the one-step look-ahead procedure would suffice to resolve any ambiguities, but since we are dealing with streaming event data in this context, we possess no knowledge about events which will arrive in the future, preventing the execution of the look-ahead procedure. We propose and have implemented three methods to deal with this issue:

1. First, disabling the look-ahead altogether and assuming that the model is deterministic enough to handle incoming events without taking the context into account ($n = 0$ in Algorithm 6.1).
2. Second (another extreme), restarting the replay of the full trace each time an event is added, thus allowing the replayer to revise earlier decisions ($n = |\sigma^L|$ in Algorithm 6.1). Note however that the replayer is configured such that no new violations may be introduced related to historical activities ($\neg \mathit{conforms}(e) \lor \neg r$). In practice, this means that the replayer can revise the state chain by modifying the execution of silent transitions, selecting alternative albeit *also enabled* transition mapped to a particular activity for activities which were parsed correctly, or selecting alternative disabled transition, although only for activities which were not parsed correctly (provided by function *conforms* in Algorithm 6.1).
3. The third method combines these two extremes by considering a part of the executed transition sequence as “frozen”, only allowing revisions for the last n steps.

As a last remark, note that the use of model-only moves in Algorithm 6.1 was disabled, to bring the replayer more in line with a real-time conformance analysis and monitoring context, where model-only moves are less appropriate.

Algorithm 6.1 Real-time event replay algorithm.

```
Input: $(P, T, F)$, $M_0$ % Given Petri net and initial marking
Input: $e = (id, act, time)$ % Arriving event to be replayed (checked)
Input: $\sigma^L$ % Trace of log activities having occurred so far for instance $id(e)$
Input: $\sigma$ % Trace of model transitions being executed so far for instance $id(e)$
Input: $m$ % Current marking of the model for instance $id(e)$
Input: $conforms : \sigma^L \rightarrow \{True, False\}$ % Function denoting conf. outcome previous event
Input: $enabled : (T \times M) \rightarrow \{True, False\}$ % Function denoting if a transition is enabled under a given marking
    (M is set of all possible markings)
Input: $nextmarking : (T \times M) \rightarrow M$ % Function returning the marking after (force) firing a given transition in
    a given marking
Input: $random : (T' \subseteq T) \rightarrow T$ % Function returning random transition from a given set of transitions
Input: $\mu : T \rightarrow A_L \cup (s, b)$ % Mapping function between model and log
Input: $n := 0$ % Number of steps to revise in historic trace (default: 0, i.e. none)
Input: $r := False$ % Denoting whether event being replayed is a revised historic event
Input: $e_n := \emptyset$ % Next incoming event (optional; used when revising earlier decisions)
Output: Executed transition $t$
1: function ReplayEvent($SN$, $e$, $e_n$, $\sigma^L$, $\sigma$, $m$, $n$, $r$)
2:     % Handle revision of historic decision
3:     if $\neg r \wedge n > 0$ then
4:         % Remove last $n$ items from $\sigma$ and revert marking $m$ to earlier state
5:         $\sigma := \langle \sigma_1, \ldots, \sigma_{|\sigma|-n} \rangle$, $m := m_{earlier}$
6:         for all $i \in \langle (n-1), \ldots, 0 \rangle$ do
7:             ReplayEvent($SN$, $\sigma^L_{|\sigma^L|-i}$, $\sigma^L_{|\sigma^L|-i+1}$, $\sigma^L$, $\sigma$, $m$, $n$, True) % Revise history
8:         end for
9:     end if
10:    % Construct candidate transition sets
11:    $MT := \{t \in T | \mu(t) = act(e)\}$, $ET := \{t \in T | enabled(t, m)\}$
12:    $IT := \{t \in T | \mu(t) = a_i\}$
13:    $CET := \{t \in T | \exists t_n \in T : [\mu(t_n) = act(e) \wedge enabled(t_n, nextmarking(t, m))]\}$
14:    $NET := \{t \in T | \exists t_n \in T : [\mu(t_n) = act(e_n) \wedge enabled(t_n, nextmarking(t, m))]\}$
15:    % Determine transition to fire
16:    if $|MT \cap ET \cap NET| > 0$ then
17:        $t := random(MT \cap ET \cap NET)$
18:    else if $|MT \cap ET| > 0$ then
19:        $t := random(MT \cap ET)$
20:    else if $|IT \cap ET \cap CET| > 0$ then
21:        $t := random(IT \cap ET \cap CET)$
22:    else if $|MT \cap NET| > 0$ then
23:        $t := random(MT \cap NET)$
24:    else if $|MT| > 0$ then
25:        $t := random(MT)$
26:    else if $|IT \cap CET| > 0$ then
27:        $t := random(IT \cap CET)$
28:    end if
29:    $\sigma := \langle \sigma, t \rangle$, $m := nextmarking(t, m)$
30:    if $t \in IT$ then
31:        $ET := \{t \in T | enabled(t, m)\}$
32:        $t := random(MT \cap ET)$
33:        $m := nextmarking(t, m)$
34:    end if
35:    if $\neg r$ then
36:        $\sigma^L := \langle \sigma^L, e \rangle$
37:    end if
38:    return $t$
39: end function
```

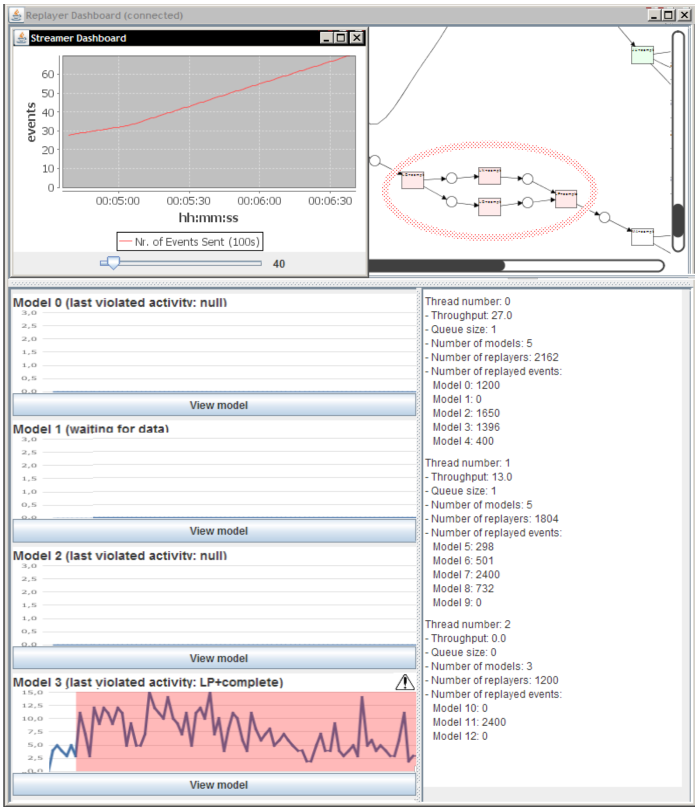
As a third remark, recall that it was mentioned in Subsection 6.2.1 that one of the drawbacks of "token game"-based replayers entails the possible creation of superfluous tokens, enabling subsequently for too much behavior. However, as was mentioned in the description of Phase 1, we note that the decomposition of a process model restricts the possible pernicious effects of the heuristics decisions taken during the conformance analysis, as each model is now limited to dealing with a smaller subset of behavior. In addition, as superfluous tokens are created following the forced firing of violating activities, the process instance or model fragment at hand is likely to be immediately indicated as "dubious" at this point, lowering the trustfulness of following events within this instance of model fragment, independent of the replay strategy being applied. In addition, recall that we are applying a hierarchical decomposition strategy, so that it is possible to perform the actual replay at a lower-granularity level than the visualization and reporting.

6.3.4 Phase 4: Reporting and Visualization

The final phase consists of reporting and visualization. Remark that, naturally, these actions can be performed while the actual conformance analysis is running. In general, two ways of result follow-up are supported by our architecture. The first one consists of the logging of various statistics by the running worker threads and replayers, which is polled regularly by decoupled components (e.g. a real-time dashboard or perhaps logged to a persistent data store). The second manner by which results can be interpreted consists of the definitions of various triggers which are to be fired once certain criteria are met, such as a model fragment overshooting a certain error rate threshold, for instance, of a high-risk activity or model fragment being violated. The actions which can be undertaken as a result are self-explanatory, e.g. sending warnings, or halting running process instances or even the complete system.

6.3.5 Implementation

Figure 6.3 depicts a screen capture of our developed proof-of-concept implementation is shown. In the prototype, events are streamed over a network in real-time using a separate program (shown as the top left window in Figure 6.3), which are received by the conformance analyzer and verified against the decomposed


Replayer Dashboard (connected)
Streamer Dashboard
events
hh:mm:ss
Nr. of Events Sent (100s)
40
Model 0 (last violated activity: null)
View model
Model 1 (waiting for data)
View model
Model 2 (last violated activity: null)
View model
Model 3 (last violated activity: LP+complete)
View model

Thread number: 0
- Throughput: 27.0
- Queue size: 1
- Number of models: 5
- Number of replayers: 2162
- Number of replayed events:
  Model 0: 1200
  Model 1: 0
  Model 2: 1650
  Model 3: 1396
  Model 4: 400

Thread number: 1
- Throughput: 13.0
- Queue size: 1
- Number of models: 5
- Number of replayers: 1804
- Number of replayed events:
  Model 5: 298
  Model 6: 501
  Model 7: 2400
  Model 8: 732
  Model 9: 0

Thread number: 2
- Throughput: 0.0
- Queue size: 0
- Number of models: 3
- Number of replayers: 1200
- Number of replayed events:
  Model 10: 0
  Model 11: 2400
  Model 12: 0

Figure 6.3: Screen capture of the developed real-time event conformance analysis prototype. A global overview of the model being checked against, error rates per submodel, and general statistics are reported. Our real-time approach allows to immediately react once a certain (user-configurable) criteria are triggered, such as model fragments (or specific activities) reaching a certain failure threshold.

model fragments. The top panel displays a global overview of the model being checked against, with violating parts highlighted. Since the analysis is performed on the basis of the decomposed model fragments, it is more straightforward to pinpoint errors to a localized area within the global view than when using the full model as-is to perform conformance analysis. The lower left panels depict error monitors per submodel, showing the error rate for each model fragment over time. The panel on the right shows general statistics and program information. Note that this real-time approach allows to immediately react once a certain (user-configurable) criteria are triggered, such as model fragments (or specific activities) reaching a certain failure threshold., and shows the error rate for each model fragment together with a global overview for the complete process model.

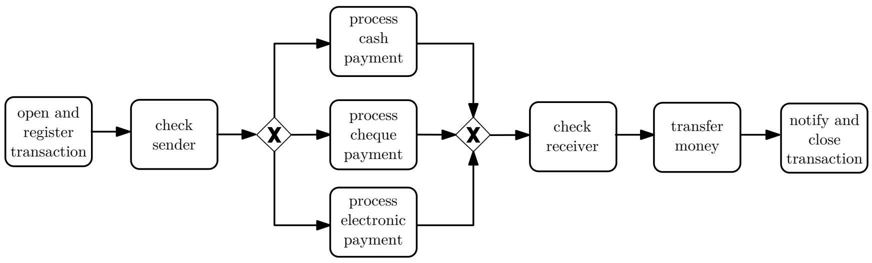


Figure 6.4: High level overview of the running example process, structured in subprocesses.

6.4 Case Example

In this section we propose the study of a realistic process case example in order to illustrate the approach presented in this chapter and its benefits.

6.4.1 Description

The modeled process describes a realistic transaction process within a banking context. The process contains all sort of monetary checks, authority notifications, and logging mechanisms responding to the new degree of responsibility and accountability that current economic environments demand. The process is structured as follows (Figure 6.4 shows a high-level overview of the complete process): it is initiated when a new transaction is requested, opening a new instance in the system and registering all the components involved. The second step is to run a check on the person (or entity) origin of the monetary transaction. Then, the actual payment is processed differently, depending of the payment modality chosen by the sender (cash, cheque and payment). Later, the receiver is checked and the money is transferred. Finally, the process ends registering the information, notifying it to the required actors and authorities, and emitting the corresponding receipt.

The process has been modeled in terms of a Petri net. The decomposition techniques based on SESE-components (see Section 6.3) is used to decompose the overall model into suprocesses. In particular, a valid decomposition where components have size at most 60 is derived. Finally, the decomposition is post-processed by merging some of the SESE-components in order to reach the final decomposition shown in Figure 6.5 (which depicts the full process): eight of the proposed subnets correspond with the eight subprocesses identified in Figure 6.4 (represented within gray rectangles), and the ninth subnet contains all the trivial connections between suprocesses (represented outside the rectangles).

6.4.2 Experimental Scenario Evaluation

To illustrate the benefits of the technique, we present two possible scenarios within the case example process.

6.4.2.1 *Scenario 1: Serial Number Check*

The modeled process defines that, whenever a client executes the payment in cash, the serial numbers must be checked (see Figure 6.4). The banking regulation states that serial numbers must be compared with an external database governed by a recognized international authority ("Check Authority Serial Numbers CASN"). In addition, the bank of the case example decided to incorporate two complementary checks to its policy: an internal bank check ("Check Bank Serial Numbers CBSN"), and a check among the databases of the bank consortium this bank belongs to ("Check Inter-Bank Serial Numbers CIBSN"). At a given point, due to technical reasons (e.g., peak hour network congestion, malfunction of the software, deliberated blocking attack, etc.), the external check CASN is not longer performed, contradicting the modeled process, i.e., all the running instances of the process involving cash payment can proceed without the required check. Using the proposed approach, this situation is detected immediately, identifying the anomalous subprocess (process cash payment), focusing the conformance analysis on it, and eventually taking the necessary countermeasures. The consequences of detecting such cases only in forensic analysis performed months after the incident are severe and difficult to recover from. The situation is depicted in Figure 6.6.

6.4.2.2 *Scenario 2: Receiver Preliminary Profiling*

During the check receiver stage, the model establishes two steps to be performed sequentially: first, a preliminary profiling analysis ("Start Receiver Pre Profiling SRPP") is executed over the receiver in order to evaluate and establish its potential risk ("Evaluate Pre Profiling EPP"). Only then, a complete background check

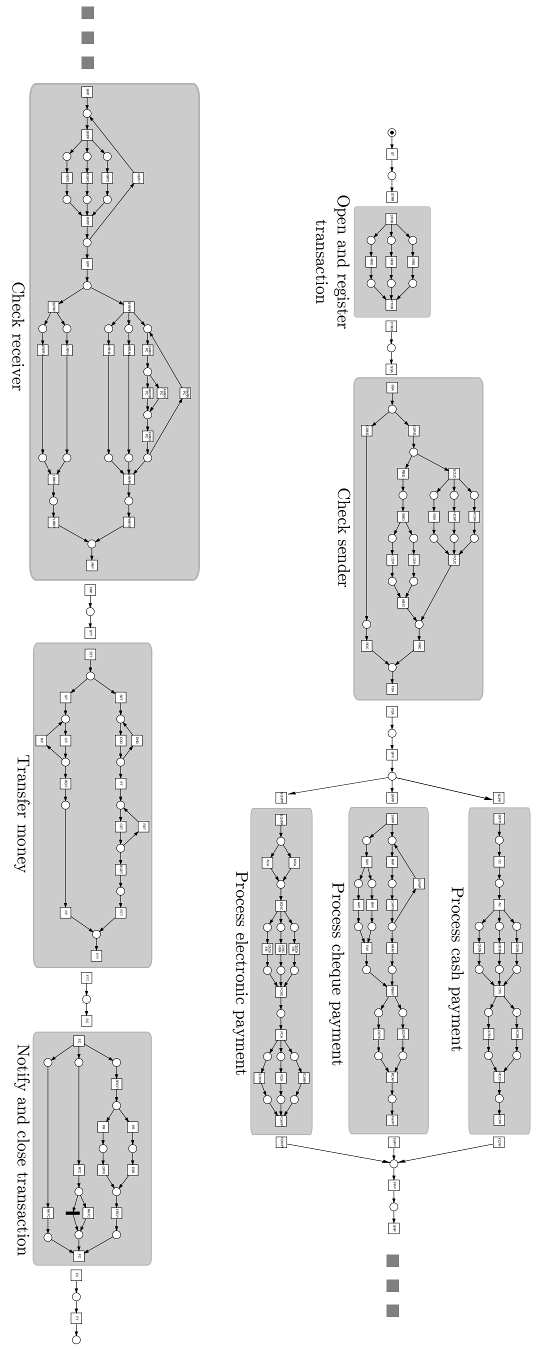
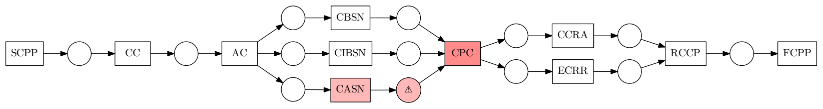

Check receiver
Transfer money
Notify and close transaction
Open and register transaction
Check sender
Process electronic payment
Process cheque payment
Process cash payment

Figure 6.5: Running example: final valid SESE-decomposition. The substructures are named according to Figure 6.4.



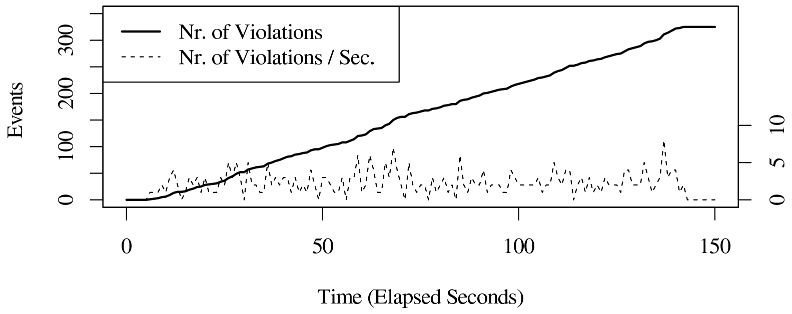


Figure 6.6: In the first scenario, the Check Authority Serial Number (CASN) activity is skipped for some process instances, causing the CPC activity to fail, due to a missing input token which was expected to be present and placed there by the execution of CASN. The figure depicts the error localized in the affected model fragment; the graph depicts the cumulative and running amount of violations detected within this fragment.

is performed over the receiver, where this check can either be more casual ("Start Low Risk Receiver Processing SLRRP ") or thoroughly ("Start High Risk Receiver Processing SHRRP") depending on the potential risk detected on the preliminary profiling. However, the presence of an inexperienced bank employee, malevolence, or simply a bad implemented bank evaluation protocol, could result in evaluating the receiver with an unfinished preliminary profile check. The situation is depicted in Figure 6.7.

6.4.3 Experimental Comparison

To benchmark the performance of our developed real-time conformance analysis technique against related approaches, a fitting event log was generated (based on the model depicted in Figure 6.5) containing ten thousand process instances (678864 events). A non-conforming ("noisy") variant of this event log was produced by inducing noise (inserting, deleting, and swapping of events) so that 10% of the included events are erroneous.

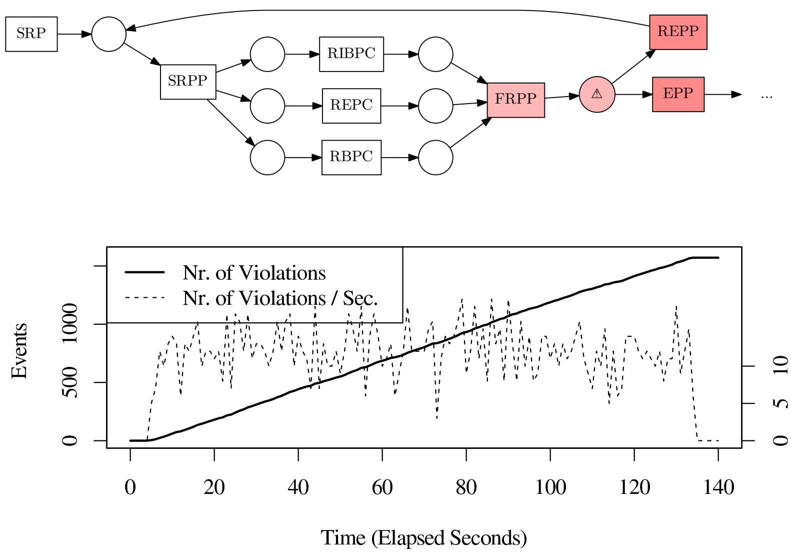


Figure 6.7: In the second scenario, the preliminary profile check for receivers is skipped (SRPP to FRPP), causing either the REPP or EPP activities to fail. The figure depicts the error localized in the affected model fragment; the graph depicts the cumulative and running amount of violations detected within this fragment.

We compare our proposed technique with the alignment based replay technique by Adriansyah et al. [108–112] as well our original implementation of the token-game based heuristic replayer (see Chapter 3). Both the non-decomposed and decomposed variants of these techniques were included, applying hereto the methodology as described in [182].

Figure 6.8 depicts the performance results of the experiment, showing the amount of time taken (x-axis) to check the conformance of the included event logs (the y-axis represents the cumulative ratio of event checks performed). As can be seen, our proposed real-time conformance analysis technique performs competitively with respect to related techniques. During the experimental run, a maximum throughput rate (number of events checked per second) was reached at 35000 with the experiment running on a single consumer laptop with three worker threads. Some additional remarks should be provided however when interpreting Figure 6.8. First, note that our proposed technique performs a *conformance check on an event-granular basis*, whereas the other techniques do so on a trace-granular level (i.e. a complete trace should be provided to perform the replay procedure). However, the event log is of sufficient size so that a step-wise

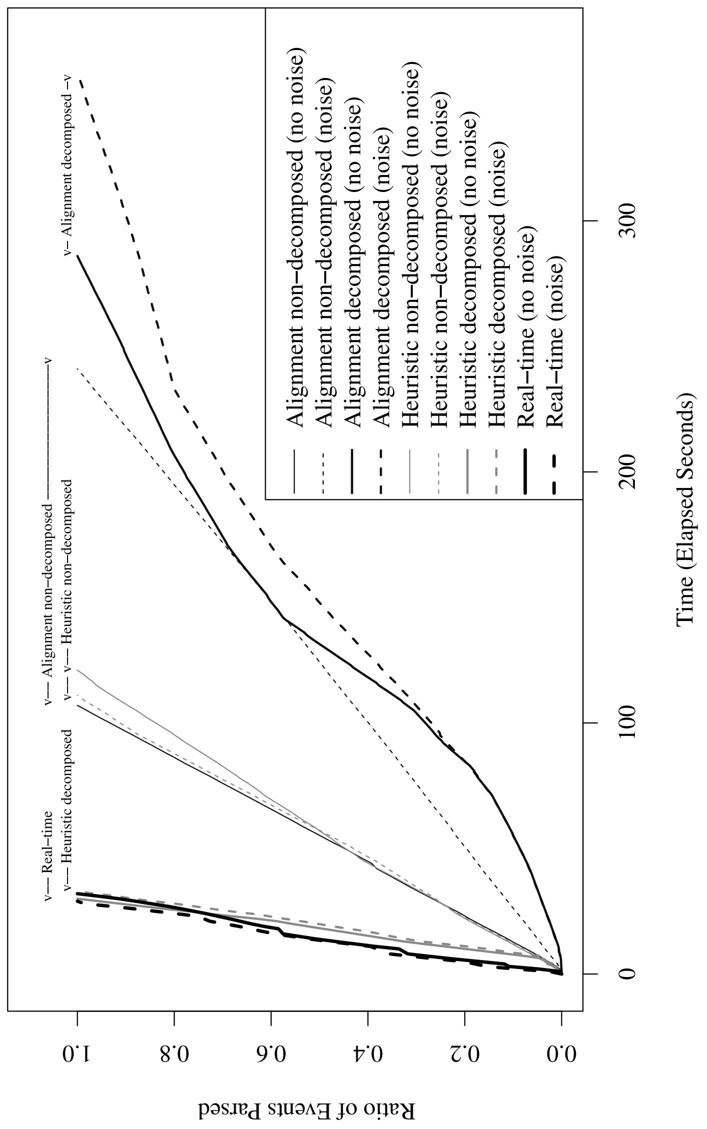


Figure 6.8: Comparison of replay performance for the included techniques in the experimental setup, showing the time taken per technique to replay the given event log.

effect is not apparent in Figure 6.8. Second, the replay procedure of the existing techniques was modified such that *each trace is checked independently* of the log context, meaning that no distinct trace grouping is performed over the log and each trace is checked as if it were belonging to an event log containing only this trace, so as to better assess the performance of these techniques in a real-time scenario (where the complete trace and log are unknown as events are arriving), rather than a post-hoc scenario where the complete event log is provided as-is. Note that—for the alignment based technique—this causes the non-decomposed version to perform better than the decomposed one. This is a perhaps unexpected result, but is caused by the fact that the alignment based techniques are geared towards checking—and as such expect—event logs as a whole. We thus emphasize the fact that these techniques have—currently—not been optimized to be applied in a real-time scenario (with an event stream being checked instead of an historical log).

6.5 Conclusions and Future Work

In this chapter, we have presented a novel business process conformance analysis technique which is able to support real-time monitoring of event-based data streams. Our approach offers a number of novel contributions, most notably a speed-up compared to related techniques, the ability to localize discrepancies and allowing real-time monitoring and thus rapid response times in mission-critical or high-risk environments, which is a significant benefit compared to existing conformance checking techniques which mainly work in an offline manner. Possible future lines of research include: streamlining visualization and reporting capabilities of our prototype, incorporating other decomposition and replay strategies, and adapting the framework into a distributed implementation, where different replayer engines run on separate machines.

Chapter 7

Explaining Clustered Process Instances

"I think you can have ten thousand explanations for failure, but no good explanation for success."
– Paulo Coelho

7.1 Introduction

This chapter describes SECPI (Search for Explanations of Clusters of Process Instances), an algorithm that is capable of finding a minimal set of control-flow characteristics for a clustered process instance.

Partitioning event logs into multiple groups of process instances is a convenient recipe for addressing the challenge of dealing with complex event logs, i.e. logs presenting a large amount of distinct process behaviour. In the literature, several trace clustering techniques have been described [62, 186–193] that are capable of intelligently splitting up an event log into multiple groups of instances so that process discovery techniques can be applied to subsets of behavior, with more accurate and comprehensible discovered models as a result. However, the application potential of trace clustering techniques is somewhat hampered by the low level of human understandability. Concretely, there exist two major problems regarding trace clustering solutions. First of all, it is a non-trivial question to find out what the driving elements are that determine a clustering technique to split up the event log in a particular way. This is because most trace clustering techniques operate at a higher level of abstraction which makes that, for instance, the concept of distance between traces is not very insightful as a means to describing a clustering solution. Secondly, end users would like to be able to understand the differentiating characteristics between multiple clusters of process instances, preferably from a domain perspective, i.e. relying on control-flow characteristics that are present in the context of the process at hand.

A posteriori comprehension of a clustering solution plays a vital role for the usefulness of separating an event log into multiple subgroups. More specifically, process analysts should be able to understand which factors determine the delineation of the discovered clusters in order to be able to give an interpretation to the solution. Currently available trace clustering techniques often lack the capability to provide insight into how a certain clustering solution is composed. Therefore, this chapter presents a new technique which allows to find explanations that describe which control-flow characteristics of a certain process instance make that this instance pertains to a certain cluster. In the remainder of this chapter, it is argued that instance-level explanations can overcome drawbacks of potential alternative explanation techniques, such as for example the visual analysis of the underlying process models. The novel technique, implemented as a plugin in ProM, is inspired by the work of Martens and Provost [194, 195], who put forward an approach for explaining text document classifications. In the context of document classification, one is often confronted with limited comprehensibility of the predictive model, even despite using so-called white box techniques such as decision trees or logistic regression, which is mainly due to the high dimensionality. Similarly, such high dimensionality comes into play when characterizing process instances by means of binary vectors representing control-flow characteristics.

Against this background, the contribution we present in this chapter is SECPI (Search for Explanations of Clusters of Process Instances), an algorithm that is capable of finding a minimal set of control-flow characteristics for a process instance, such that if these characteristics were not present, the process instance would not remain within its current cluster. Furthermore, the implementation allows to visualize explanations in the respective process models so that users can easily observe what characteristics make that a process instance belongs to a certain cluster. In the experimental evaluation, it is shown that our technique strikes a better balance between accuracy and comprehensibility, as compared to conventional white box classification models. The technique presented in this chapter is not only valuable for providing a solution for interpreting trace clustering solutions, SECPI can be advantageous in the case of domain-relevant splits of an event log. More specifically, because the technique is capable of finding discriminating factors between groups of process instances, we can discover control-flow explanations for groups of process instances as defined by some exogenous criterion. For instance, we might investigate what control-flow characteristics discriminate between cases that were completed in time versus cases that transgressed the allowed time frame. Another example case is finding explanations for cases going over the allowed budget vs. cases that remain within budget.

7.2 Trace Clustering

Trace clustering is an interesting approach to deal with the problem that many event logs contain an extensive amount of distinct behavior (i.e. process variants), because it allows the user to split up a log so that multiple distinct models can be learned to describe the underlying business process.

7.2.1 State of the Art

Table 7.1 provides an overview of the most important currently available trace clustering techniques. Several currently available trace clustering techniques translate the learning problem into a propositional setting by converting an event log into an attribute-value data set, to which well-known clustering techniques such as k-means or hierarchical clustering can be applied. The pioneering study by Greco et al. [62] proposes a projection of the traces onto a set of suitable features, in this case a set of frequent k-grams of activity sequences. The projection of an event log onto a feature set was further explored by Song et al. [186]. Their trace clustering implementation allows for a multitude of profiles to determine the vector associated with each process instance. As such, they define activity, transition, performance, case attribute profiles, etc. Furthermore, whereas Greco et al. relied on k-means, Song et al. provide a full range of distance metrics and clustering algorithms. Recently, the use of dimensionality reduction techniques was investigated in the context of profile-based trace clustering [192]. Along the same line of reasoning, Bose and van der Aalst [189] propose two additional trace clustering techniques. In [196], the authors refine the technique of Greco et al. [62] by making use of so-called conserved patterns (e.g. Maximal, Super Maximal, and Near Super Maximal Repeats) as projection features. The implementation applies hierarchical clustering instead of k-means. In [188], Bose and van

Table 7.1: Available trace clustering techniques and their characteristics.

Author	Data Representation	Clustering Technique	Clustering Bias
Greco et al. [62]	propositional	k-means	instance similarity: alphabet k-grams
Song et al. [186]	propositional	various	instance similarity: profiles
Ferreira et al. [187]	event log	first order Markov mixture model by EM	maximum likelihood
Bose and van der Aalst [189]	bag of strings	hierarchical clustering	instance similarity: string edit distance
Bose and van der Aalst [196]	propositional	hierarchical clustering	instance similarity: conserved patterns
Folino et al. [190]	event log	enhanced Markov cluster model	maximum likelihood
De Weerdt et al. [191]	event log	active model-driven clustering	combined process model fitness
Ekanayake et al. [193]	propositional	various	similarity and model complexity

der Aalst turn away from the use of an attribute-value feature space, but instead regard log traces as strings over the alphabet of task labels and as such calculate the distance between traces directly by exploiting the idea of a string edit distance. However, in contrast to contemporary approaches such as the Levenshtein distance [197], specific substitution, insertion and deletion costs are derived so as to take into account characteristic features of an event log, such as concurrency of tasks. A second class of trace clustering techniques can be best described as model-driven. Inspired by the work of Cadez et al. [198] in the area of web usage mining, Ferreira et al. [187] propose to cluster sequences by learning a mixture of first-order Markov models using the Expectation-Maximization (EM) algorithm. This technique has been applied in [199] to server logs. Also in [190], Folino et al. make use of Markov chain clustering. Finally, De Weerdt et al. [191] propose the ActiTraC-algorithm, which strives to optimize the combined fitness of the underlying process models.

7.2.2 Problem Statement

As indicated in the introduction, the problem with existing trace clustering techniques is that they provide little to no insight into the actual reasoning of partitioning an event log in a particular way. From a model learning perspective, the clustering bias of a trace clustering technique determines how a solution is constructed. As shown in Table 7.1, the clustering techniques described in the process mining literature employ a wide variety of clustering biases. On the one hand, a subset of techniques relies on the concept of distance as a measure of instance similarity. Model-driven techniques on the other hand rely on maximum likelihood or fitness optimization. Observe that the ex-post, aggregated fitness of the underlying models is an often employed quality measure for trace clustering solutions, see [188, 191].

For distance-based clustering, typical data mining techniques such as k-means or hierarchical clustering are applied. As such, the distance itself is a potential candidate for explaining a clustering result. For instance, one could visualize the instances in a networked graph or make use of comparative statistical analysis of the underlying variables that determine the inter and intra-cluster distances. However, a projection of process instances onto process features will typically generate a large amount of variables (e.g. the combined number of 2- and 3-grams for a set of twenty labels is 8400), which seriously complicates such an approach. To this, it should be added that due to the large amount of variables, distance-based techniques suffer from the curse of dimensionality problem [200]. As described in [201], conventional proximity metrics in high-dimensional space may not be qualitatively meaningful. Therefore, it is argued that the value of the distance concept for providing human understanding to a trace clustering solution is low. As for model-driven techniques, the natural explanation method is a visual analysis of the resulting cluster models. In the next section, it is explained why a visual analysis of the underlying process models of a clustering result is often insufficient.

Moreover, from a user's perspective, one would like to explain differences between clusters according to context-specific process characteristics. Distances between processes instances or basic comparative statistics are not capable of doing so. Despite their distinct clustering biases, these techniques are not capable to indicate what characteristics differ among clusters, nor do they show why instances are part of a certain cluster or not. Typically, the concept of a "distance" provides the user with some insight regarding the formed clusters. For vectors representing process traces, the concept of distance is much less intuitive because it always relies on some choice of features and often the set of features is large. Another often employed approach to assess the differences between clusters is the computation of simple comparative statistics of the different variables. For trace clustering techniques operating at a propositional level, such an approach might provide some insight into the cluster differences. Nevertheless, typically thousands of features are considered to "propositionalize" an event log. Accordingly, it seems impossible that such a simple method could actually describe differences between clusters very well. To this, it should be added that multiple clustering algorithms don't use a projection onto a vector space which basically means that you don't have actual variables to compare clusters with. So either because of the nature of the clustering algorithm or because of an explosion of the number of features, it is a challenge to find out the key differentiating factors between clusters of process instances.

7.3 Alternative Cluster Explanation Techniques

Before outlining our instance-level cluster explanation approach in the next section, six alternative techniques, potentially useful for cluster explanation, are explored. These approaches can be described as global explanation methods because they seek to provide human understanding from a general viewpoint, i.e. aggregating over the entire set of process instances.

7.3.1 Visual Analysis of the Process Models

The most obvious method for obtaining human understanding of a trace clustering result is a visual comparative analysis of the underlying discovered process models. However, several drawbacks exist regarding such an approach. First of all, identifying dissimilarities between two process models often requires a high level of expertise, and is even further complicated by the fact that differences have to be identified for a set of process models. In addition, since these models are discovered, they might still remain large and complex, thus difficult to interpret. Second, process discovery techniques generalize by making a trade-off between recall, precision, generalization and simplicity and accordingly do not always represent the behavior in the cluster perfectly, which might further obfuscate differences between clusters. Moreover, the trade-off between quality dimensions differs according to the process discovery algorithm used, so that the resulting behavior might vary depending on the chosen technique to learn a visual model. Finally, because these models stem from the same process context, differences might reside in very subtle elements, difficult to identify visually. Therefore, the visual inspection of the underlying discovered process models will often be a time-consuming and inaccurate method for cluster explanation.

7.3.2 Process Model Similarity Metrics

A second solution for cluster characterization is the use of automated similarity analysis techniques for business process models. There exists an extensive body of literature on the behavioral equivalence of process models [202, 203]. However, common notions such as trace equivalence and (branching) bisimulation [204] only provide the user with a strict true or false answer, as pointed out in [205], which makes them inapplicable to our context. Other researchers have proposed more fine-grained similarity metrics for comparing business process models, which potentially suit better the problem at hand. For instance, Alves de Medeiros et al. [206] propose a recall and precision metric to quantify process model similarity based on observed behavior. Yet, several factors affect the suitability in the context of cluster explanation negatively. First, perfect precision and recall values, i.e. both equaling 1, do not guarantee behavioral equivalence. It is also unsure which common event log to use for comparison, because the original complete log will not fit the models underlying the clusters well. Observe that imperfect fitness and a mismatch between tasks in the log and in the model will severely impact the interpretation of the precision and recall values. Finally, the technique has two more general disadvantages, which also applies to other process model similarity quantification approaches (e.g. [207]): first, they are not capable of providing insight into the root causes of dissimilarity and second, the user is faced with an increased comparative complexity as soon as more than two clusters are present in a solution as each cluster should be compared separately to every other cluster.

7.3.3 Automated Dissimilarity Visualization in Process Models

As a third potential alternative cluster explanation technique, Dijkman proposed a process similarity technique that is capable of returning the exact positions of differences between two process models [208] by means of typology of process model differences defined in [209]. This technique has the advantage to be capable of identifying root causes of process model differences. Observe however that the technique is designed to diagnose Event-Driven Process Chains (EPCs). Furthermore, the technique has exponential complexity and can only compare two models at a time.

7.3.4 Footprints and Behavioral Profiles

The behavioral similarity metric in [207] is founded on the concept of causal footprints [210]. Despite the fact that the metric itself is not useful for our purpose, the technique of footprint comparison is a potential candidate for finding trace cluster explanations. It can be regarded as a technique to abstract away from interpreting and comparing control-flow behavior of process models, by means of providing insight into causal relations. As such, they might help a user to identify differences more easily. For model-based footprints, the issue of abstraction by discovery remains: the discovered model might obfuscate differences between clusters. Other researchers have proposed similar concepts to compare process models, most notably behavioral profiles [103] and transition adjacency relations [211]. It is pointed out that the abstraction by discovery issue might be resolved by deriving footprints from the clustered logs instead of the process models. For instance, dissimilarities can be visualized by aggregating all instance-level profiles of two clusters into a log-level matrix for each cluster. Because this is a data first approach, no discovered model is required to construct such matrices. However, with a growing activity alphabet, these matrices can quickly become difficult to interpret. In addition, the comparison of more than two clusters becomes unwieldy as well, requiring a batch-style pairwise analysis. This might seriously complicate the task of explaining clusters of process instances.

7.3.5 White Box Classification Model Learning

As a fifth potential cluster explanation technique, white box classification techniques could be employed. In fact, this approach corresponds to a reverse engineering of conventional trace clustering techniques where process instances are projected onto feature vectors. One could apply rule learning or decision tree algorithms to a data set obtained by combining the feature vectors and the cluster identifier. As such, a classification model can be learned that explains why process instances belong to a certain cluster. White box classification techniques such as RIPPER [132] and C4.5 decision trees [212] have been shown to be useful in the process mining domain, e.g. in [29, 213]. However, the usefulness of such classification models is determined by two important parameters. First of all, the model should present a high classification accuracy so as to be valid. Furthermore, the model should be comprehensible for end users. In Section 7.5, an experiment is presented comparing white box classification techniques to our newly proposed algorithm.

7.3.6 Cross-Cluster Conformance Checking

The last potential alternative technique for providing human understanding to a trace clustering solution is the use of conformance checking. In this particular case, one could rely on cross-cluster model-log alignment for identifying differences between clusters. By aligning or replaying an entire cluster log with the different process models underlying the other clusters in the clustering solution, it is feasible to visualize differences at a global level. However, cross-cluster conformance checking can also be performed on an instance-level basis, investigating the different alignments or replay outcomes of one particular instance with the discovered process models underlying the other clusters. Although both the use of instance-level as well as global cross-cluster alignments has the potential to reveal differences between clusters of process instances, one notable downside is the mandatory use of the discovered process models for finding explanations. Our novel approach avoids the use of the underlying discovered process models and solely relies on the data (i.e. the process instances) in the clustered logs. Observe that pure trace alignment [214] also avoids the use of models, however, it is unsure how this particular technique can be applied to multiple event logs at once.

7.4 Instance-Level Explanations with SECPI

We present a completely new analysis approach for explaining the differences between clusters of process instances, overcoming many of the drawbacks of alternative techniques. The basic idea is shown in Figure 7.1. Instead of providing a global explanation, concise if-then rules are learned for each individual instance, with a conjunction of control-flow characteristics (e.g. "sometimes directly follows"-relations) forming the antecedent and the cluster switch as consequence. As such, an explanation is a rule that stipulates which characteristics are the determining factors that make that a certain instance pertains to its current cluster. Experiments presented in the next section show that we can learn accurate and concise explanations.

7.4.1 Constructing the Data Set

First, process instances are converted into feature vectors. The implementation supports several attribute templates (e.g. activity presence, always/sometimes

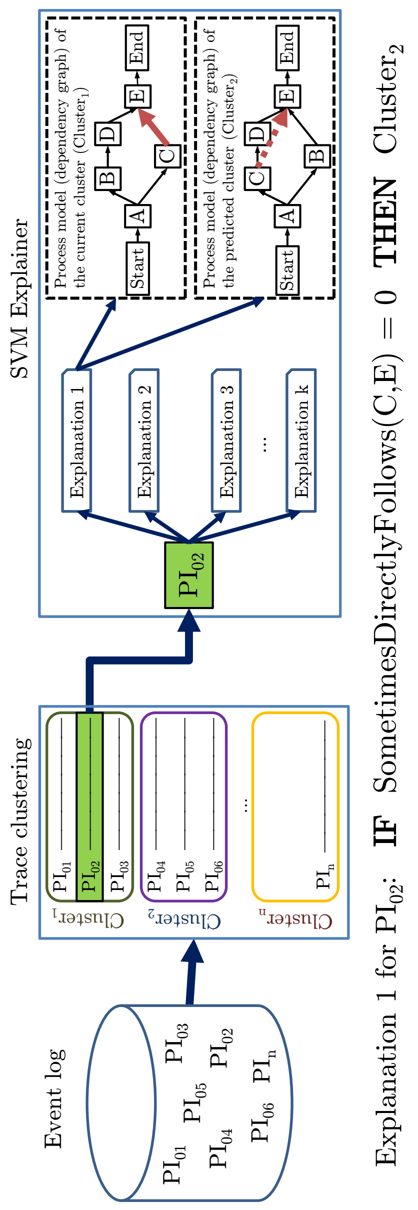

Event log
PI01
PI04
PI05
PI03
PI06
PI02
PIn
Trace clustering
Cluster1
Cluster2
Clustern
PI01
PI02
PI03
PI04
PI05
PI06
...
PIn
SVM Explainer
PI02
Explanation 1
Explanation 2
Explanation 3
...
Explanation k
Process model (dependency graph) of the current cluster (Cluster1)
Start
A
B
C
D
E
End
Process model (dependency graph) of the predicted cluster (Cluster2)
Start
A
B
C
D
E
End
Explanation 1 for PI02: IF SometimesDirectlyFollows(C,E) = 0 THEN Cluster2

Figure 7.1: Overview of SECPI: for each process instance (PI) in the event log, one or more explanations are learnt and ranked according to their length. An explanation is a simple if-then rule with a conjunction of characteristics (as few as possible) which should not be present (i.e. set to zero) in order for the instance to rather belong to a different cluster. The SECPI implementation is capable of visually reflecting these key determinants of cluster membership in the respective process models, as illustrated on the right hand side.

weak order relations), however our experiments show that the "sometimes directly follows" attribute template provides solid explanatory power from a control-flow perspective. The *SometimesDirectlyFollows*(a, b) attribute for two activities a and b evaluates to true when these two activities both occur in the instance (potentially multiple times) and follow each other directly at least once, and to false otherwise (never follow each other directly or do not both occur).The data set is completed by adding the appropriate cluster label to each instance. As such, a labeled data set is obtained to which supervised data mining techniques can be applied. It is pointed out that non-control-flow attributes easily can be incorporated.

7.4.2 Deriving Explanations from a Support Vector Machine (SVM) Classifier

As stated earlier, our approach is inspired by [195] in which an algorithm is proposed to find explanations for document classifications. The most important similarity is the use of an SVM-based classifier as the base model from which explanations are derived. As for document classification, SVMs are ideally suited in our context because the use of multiple or complex attribute templates will quickly lead to massive dimensionality. By employing the well-known "liblinear"-library for large-scale linear classification based on linear kernel SVMs, our approach can support data with millions of instances and features. For more details about SVMs, we refer to [215].

We adapt the approach in [195] to the context of trace clustering with some key modifications. First, support for multi-class prediction has been developed because in our context it is highly plausible to have more than two clusters. Second, we configure the algorithm in such a way that explanations can be restricted to behavior present in a process instance (only swaps from 1 to 0 are considered). Third, several performance optimizations have been introduced: we avoid considering attributes with no variability (always 0 or 1), prevent repeat checking of same attribute combinations, and consequently avoid to expand on attribute combinations that have been considered before. These improvements are explained in more detail below.

7.4.3 Algorithm SECPI (Search for Explanations for Clusters of Process Instances)

Algorithm 7.1 provides a formalized overview of the workings of the SECPI algorithm. As inputs, an instance to be explained (a process trace in a cluster) is given, defined as a sequence of binary attributes (generated using the attribute templates as discussed above). Next, a classifier is assumed to be trained over the data set which is able to, for a given feature vector, return a predicted class label and associated score (i.e. probability). Finally, three configuration options have to be set: *iterations* denotes the depth to search for explanations for the given instance. Increasing this value increases the run time but leads to more (albeit longer) explanations. The *zero_to_one* parameter denotes whether 0 to 1 attribute value swaps should be allowed. Since the instance attributes denote characteristics of the instance which are present (such as the direct following of two activities, for instance), it is recommended to set this parameter to *False*, as explanations denoting that a trace would not appear in its cluster when it did not present a specific characteristic are generally easier to interpret than explanations denoting that a trace should have a certain characteristic (as the question is then asked where and how exactly this characteristic would manifest itself within the trace). Additionally, since the multitude of all attributes for a trace are set to 0, the list of retrieved rules will be shorter and better fine-tuned to the actual behavior as seen in the process instance. Finally, *require_support* denotes whether attribute value swaps should be taken into account for attributes which are always set to 0 or 1 (i.e. no variability). Again, it is recommended to set this to a *True* value, as providing explanations which require behavior which is nowhere seen in the log are most likely less useable than those which do only incorporate seen behavior.

As output, a set of explanatory rules is returned, formalized as a set of sets of attribute indices. Each set of indices represents a candidate explanation, and should be interpreted as follows: "this process instance would leave its current cluster when all the following attributes would be inverted"—or, in case where *zero_to_one* is set to *False*: "when it would not exhibit the behavior as represented by these attributes". To construct this set of explanations, the algorithm applies a heuristic, best-first search procedure with pruning. First, each candidate single attribute is evaluated to see whether rules composed of only one attribute can be found. If swapping an attribute's value does not lead to a class change, a combination of indices (in this case a single index) is added to `E` to be expanded in the next step.

Algorithm 7.1 Formalisation of the SECPI algorithm.

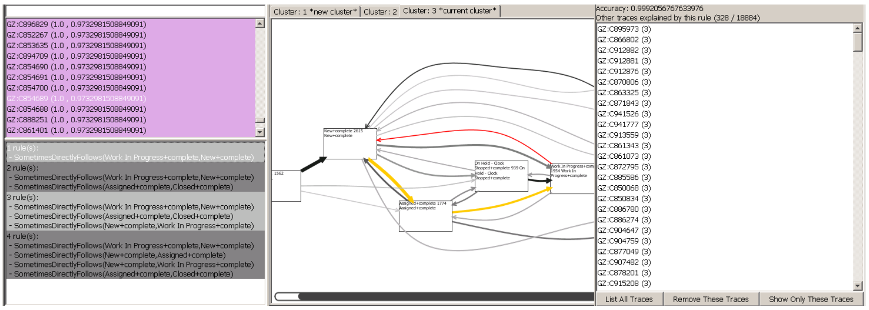
```
Input: $I := \langle I_i \in \{0,1\}, i = 1,2,\dots,|I|\rangle$ % Process instance $I \in L$ ($k$ clusters)
Input: $C : L \mapsto \{1,2,\dots,k\}$ % Trained classifier with scoring function $f_C$
Input: $iterations := 30$, $zero\_to\_one := False$, $require\_support := True$ % Configuration
Output: Set of explanatory rules $R$
1: function SECPI( $I$, $C$, $iterations$, $zero\_to\_one$, $require\_support$ )
2:     $c := C(I)$, $p := f_C(I)$ % Predicted cluster and corresponding probability
3:     $R := \{\}$ % Set of instance explanations (set of sets)
4:     $E := \{\}$ % Combinations to expand on (set of sets)
5:     % Search for single attribute explanations
6:     for all $i \in \{1, \dots |I|\}$ do
7:         if $IsAllowedSwap(I, i)$ then
8:             $I' := SwapAttributes(I, \{i\})$
9:             $c' := C(I')$ % New cluster label
10:            $p' := f_C(I')$ % New probability
11:            if $c' \neq c$ then $R := R \cup \{i\}$
12:            else $E := E \cup \{i\}$ end if
13:        end if
14:    end for
15:    % Iteratively search for multi attribute explanations
16:    for all $iteration \in \{1, \dots, iterations\}$ do
17:        $combo := argmax_{A \in E}\,(p - f_C(SwapAttributes(I, A)))$ % Best combination
18:        $combos' := \{\}$
19:        for all $i := 1 \rightarrow |I|$ do % Expand combination
20:            $combo' := combo \cup \{i\}$
21:            if $combo \neq combo' \wedge IsAllowedSwap(I, i) \wedge \neg IsSubsumed(R, combo')$ then
22:            $combos' := combos' \cup \{combo'\}$ end if
23:        end for
24:        for all $combo' \in combos'$ do
25:            $I' := SwapAttributes(I, combo')$
26:            $c' := C(I')$ % New cluster label
27:            $p' := f_C(I')$ % New probability
28:            if $c' \neq c$ then $R := R \cup combo'$
29:            else $E := E \cup combo'$ end if
30:            $E := E \setminus combo$ % Don't check this combination again
31:        end for
32:    end for
33:    return $R$
34: end function
35: function IsSubsumed($R$, $A$)
36:     % Check whether attributes with indices $\in A$ are subsumed by explanation in $R$
37:     for all $E \in R$ do
38:         if $E \in A$ then return $True$ end if
39:     end for
40:     return $False$
41: end function
42: function IsAllowedSwap($I$, $a$)
43:     % Check whether attribute with index $a$ in instance $I$ may be swapped
44:     $a' := abs(I_a - 1)$
45:     if $\neg zero\_to\_one \wedge I_a = 0$ then return $False$ end if
46:     if $require\_support \wedge \nexists J \in L : J_a = a'$ then return $False$ end if
47:     return $True$
48: end function
49: function SwapAttributes($I$, $A$)
50:     % Swap attributes with indices $\in A$ in instance $I$
51:     $I' := \langle I'_i \in \{0,1\}, i = 1,2,\dots,|I| : I'_i =$ if $i \notin A$ then $I_i$ else $abs(I_i - 1)\rangle$
52:     return $I'$
53: end function
```

Next, a number of iterations is performed as set by the *iterations* parameter. A best-first candidate selection from all currently available combinations to expand on is chosen, based on the classifier's scoring function. The goal is to first explore the set of attribute indices for which swapping their values moves the instance farthest away from its current class label (i.e. cluster). Expansions on this combination are created by creating a new set of combinations $combos'$ by adding each allowed attribute to the set of *combo*. Expansions which are equal to *combo* (i.e. the added attribute was already used in *combo*) or which are subsumed by an already existing explanation (the expansion contains all attribute indices of an existing explanation and thus adds no value) are not considered. Once all expansions are built, they are evaluated to see if they lead to a class change. Expanded combinations are removed from E to prevent them being chosen again in the next iteration.

As a classification model, we use a combination of k (the number of clusters) SVM models to allow for multi-class classification with SVMs. To retrieve the predicted class label and score, we apply a winner-takes-all strategy as follows. An SVM model is built per cluster to predict whether an instance is in-cluster (label: 1) or out-of-cluster (label: 0). To predict the label and probability of an instance, the probability that the instance is out or in their respective cluster is evaluated for all SVMs (with probability p_k if predicted in-cluster and $1 - p_k$ if predicted out-of-cluster). The SVM model with the highest probability determines the label (and its corresponding probability). Note that other classifiers (such as decision tree or rule based classifiers) could, in theory, also be applied in the SECPI algorithm as long as a scoring function can be defined, and in fact could also return small-sized instance explanations—as is our goal—even although their model itself (in terms of number of rules or decision tree nodes for example) can still be large. However, the construction of such models becomes unwieldy when dealing with high dimensional data sets, so that SVMs remain a better suitable classifier for use within our proposed technique.

7.5 Experimental Evaluation

The technique is implemented in ProM 6 as the SVMExplainer-plugin. Apart from applying the SECPI algorithm, the end user has the option to interact with a visual interface, allowing for the following analysis tasks. First, inspection of instances and their explanations (SECPI). Second, the projection of an instance



```
GZ:C896829 (1.0 , 0.9732981508849091)
GZ:C852267 (1.0 , 0.9732981508849091)
GZ:C853635 (1.0 , 0.9732981508849091)
GZ:C894709 (1.0 , 0.9732981508849091)
GZ:C854690 (1.0 , 0.9732981508849091)
GZ:C854691 (1.0 , 0.9732981508849091)
GZ:C854700 (1.0 , 0.9732981508849091)
GZ:C854689 (1.0 , 0.9732981508849091)
GZ:C854688 (1.0 , 0.9732981508849091)
GZ:C888251 (1.0 , 0.9732981508849091)
GZ:C861401 (1.0 , 0.9732981508849091)

1 rule(s):
 - SometimesDirectlyFollows(Work In Progress+complete,New+complete)
2 rule(s):
 - SometimesDirectlyFollows(Work In Progress+complete,New+complete)
 - SometimesDirectlyFollows(Assigned+complete,Closed+complete)
3 rule(s):
 - SometimesDirectlyFollows(Work In Progress+complete,New+complete)
 - SometimesDirectlyFollows(Assigned+complete,Closed+complete)
 - SometimesDirectlyFollows(New+complete,Work In Progress+complete)
4 rule(s):
 - SometimesDirectlyFollows(Work In Progress+complete,New+complete)
 - SometimesDirectlyFollows(New+complete,Assigned+complete)
 - SometimesDirectlyFollows(New+complete,Work In Progress+complete)
 - SometimesDirectlyFollows(Assigned+complete,Closed+complete)
```

```
Accuracy: 0.9992056767633976
Other traces explained by this rule (328 / 18884)
GZ:C895973 (3)
GZ:C866802 (3)
GZ:C912882 (3)
GZ:C912881 (3)
GZ:C912876 (3)
GZ:C870806 (3)
GZ:C863325 (3)
GZ:C871843 (3)
GZ:C941526 (3)
GZ:C941777 (3)
GZ:C913559 (3)
GZ:C861343 (3)
GZ:C861073 (3)
GZ:C872795 (3)
GZ:C885586 (3)
GZ:C850068 (3)
GZ:C850834 (3)
GZ:C886780 (3)
GZ:C886274 (3)
GZ:C904647 (3)
GZ:C904759 (3)
GZ:C877049 (3)
GZ:C907482 (3)
GZ:C878201 (3)
GZ:C915208 (3)
```

Figure 7.2: Screenshot of implemented SECPI plugin.

and its explanation onto a visual (discovered) process model (built per cluster) to show where the characteristics contained in the explanation are located. Note that the feasibility of this projection depends on the attribute templates chosen and is therefore purely optional. Attribute templates can be modified or added according to the requirements of analysts, even when no simple strategy for projecting them onto a visual model exists. As a third analysis option, we have added the capability to aggregate instance level explanations ("show all instances which can be explained by this rule") in order to search for a minimal set of rules which explain all clustered instances. As such, a good middle ground technique is provided between very detailed instance-level based inspections and the large models given by global white box classifiers.

7.5.1 Experimental setup

In order to study the explanatory power of SECPI, an experimental evaluation is performed with six real-life logs (stemming from various sectors and workflow systems). Table 7.2 provides an overview of the basic characteristics of the included logs, i.e. the number of instances, number of distinct instance variants, the number of activities and the average number of activities per instance.

We evaluate the performance of SECPI in comparison with two white box global classifiers: RIPPER [132] and C4.5 decision trees [212], as implemented in the Weka data mining framework. We recognize that such a comparison is imperfect, however, it is deemed the most insightful approach for demonstrating the capabilities of SECPI. Observe that we employ simplification techniques [216] for decision trees and rule sets and even consider variants with increased pruning in order to maximize the comparative fairness of our experiments. For the creation

Table 7.2: Event logs used in experimental setup with their characteristics.

Event Log	Instances	Instance Variables	Activities	Average Activities per Instance
incident	24770	1174	18	5.01
purchase	10487	76	23	9.33
telecom	17812	1908	42	4.68
outsource	18884	390	7	3.86
docflow	12391	1411	70	5.30
incman	956	212	22	11.73

of clusters, we rely on the ActiTraC-algorithm [191] with default configuration settings. The scope of this study is narrowed down further by generating the root data set with the "sometimes directly follows" attribute template only. Finally, observe that we evaluate each technique using 3, 5, and 10 clusters. This choice is arbitrary, however, the quest for finding the optimal number of clusters is a specific and highly difficult research problem on its own. This study is not attempt to solve this problem as we merely want to shed light onto the explanatory power of SECPI under varying clustering sizes.

7.5.2 Results of comparing SECPI to white box techniques

Table 7.3 shows the results of the empirical experiment. The following metrics are reported: classification accuracy denotes the percentage of correctly classified instances based on the trained classifier (SVMs for SECPI, C4.5 or RIPPER). Observe that this accuracy metric does not refer to the quality of the clustering solution itself. Obviously, the mapping of process instances to clusters (clustering) is considered as input, and the classification accuracy reported thus refers to whether or not the supervised method used to explain the clustering is capable of mapping each process instance to the same cluster as per the original mapping of the clustering technique. Next, the ratio of explainable instances is given. In this setup, incorrectly classified instances are considered unexplainable, but remark that it is in fact well possible to inspect explanations given to incorrectly classified instances (i.e. so as to analyse why the classifier mislabeled these cases). Note that for SECPI, some cases exist for which no explanation could be retrieved, even despite the instance being correctly classified. For SECPI, note that not for all correctly classified instances an explanation could be retrieved in some cases. This is due to the parameter configuration applied: although an unlimited number of iterations were performed, the *zero_to_one* and *require_support* parameters were kept default, in line with our reasoning for limiting the use of both these

Table 7.3: Results of the experimental evaluation comparing SECPI with C4.5 and RIPPER. In general, SECPI is capable of striking a better balance between accuracy and comprehensibility, in this case represented by the classification accuracy and the average explanation length. This table shows the results for 3 clusters.

Event Log	Technique	3 Clusters					
		Acc.	Ex.I.	A.Ex.L. (StDev)	Nr.L.	T.S.	Nr.R.
incident	SVMExplainer	1.00	0.58	1.81 (1.38)	–	–	–
incident	C4.5 (default)	0.99	0.99	18.36 (6.27)	72	143	–
incident	C4.5 (increased pruning)	0.83	0.83	3.00 (0.00)	4	7	–
incident	RIPPER (default)	1.00	1.00	47.50 (11.89)	–	–	51
incident	RIPPER (increased pruning)	0.74	0.74	–	–	–	1
purchase	SVMExplainer	1.00	1.00	4.06 (0.38)	–	–	–
purchase	C4.5 (default)	1.00	1.00	3.03 (0.28)	11	21	–
purchase	C4.5 (increased pruning)	1.00	1.00	2.00 (0.00)	–	–	–
purchase	RIPPER (default)	1.00	1.00	11.98 (0.33)	–	–	8
purchase	RIPPER (increased pruning)	1.00	1.00	–	–	–	1
telecom	SVMExplainer	0.99	0.90	1.50 (0.60)	–	–	–
telecom	C4.5 (default)	0.98	0.98	64.99 (19.40)	257	513	–
telecom	C4.5 (increased pruning)	0.78	0.78	2.00 (0.00)	–	–	–
telecom	RIPPER (default)	0.99	0.99	168.25 (41.78)	–	–	166
telecom	RIPPER (increased pruning)	0.78	0.78	–	–	–	1
outsource	SVMExplainer	1.00	0.39	1.72 (1.03)	–	–	–
outsource	C4.5 (default)	1.00	1.00	11.43 (2.04)	24	47	–
outsource	C4.5 (increased pruning)	0.91	0.91	2.00 (0.00)	2	3	–
outsource	RIPPER (default)	1.00	1.00	18.75 (3.51)	–	–	2–
outsource	RIPPER (increased pruning)	0.88	0.88	–	–	–	1
docflow	SVMExplainer	0.98	0.98	1.27 (0.84)	–	–	–
docflow	C4.5 (default)	0.99	0.99	16.19 (5.30)	69	137	–
docflow	C4.5 (increased pruning)	0.86	0.86	4.17 (0.83)	7	13	–
docflow	RIPPER (default)	0.99	0.99	41.86 (12.02)	–	–	48
docflow	RIPPER (increased pruning)	0.80	0.80	2.00 (0.00)	–	–	2
incman	SVMExplainer	0.98	0.97	2.71 (1.06)	–	–	–
incman	C4.5 (default)	0.97	0.97	7.84 (2.33)	26	51	–
incman	C4.5 (increased pruning)	0.85	0.85	3.96 (0.81)	7	13	–
incman	RIPPER (default)	0.99	0.99	15.56 (5.83)	–	–	16
incman	RIPPER (increased pruning)	0.83	0.83	5.87 (0.33)	–	–	3

Legend: Acc.: Accuracy, Ex.I.: Number of Explanable Instances, A.Ex.L.:Average Explanation Length, Nr.L.: Number of Leaves, T.S.: Tree Size, Nr.R.: Number of Rules.

Table 7.3 (continued): Results of the experimental evaluation comparing SECPI with C4.5 and RIPPER. This table shows the results for 5 clusters.

Event Log	Technique	5 Clusters					
		Acc.	Ex.I.	A.Ex.L. (StDev)	Nr.L.	T.S.	Nr.R.
incident	SVMExplainer	0.99	0.79	1.76 (1.12)	–	–	–
incident	C4.5 (default)	0.99	0.99	23.22 (6.64)	151	30	–
incident	C4.5 (increased pruning)	0.84	0.84	5.80 (0.92)	9	17	–
incident	RIPPER (default)	0.99	0.99	72.58 (12.26)	–	–	92
incident	RIPPER (increased pruning)	0.79	0.79	6.00 (0.00)	–	–	2
purchase	SVMExplainer	1.00	1.00	4.11 (0.42)	–	–	–
purchase	C4.5 (default)	1.00	1.00	3.03 (0.35)	12	23	–
purchase	C4.5 (increased pruning)	1.00	1.00	2.00 (0.00)	–	–	–
purchase	RIPPER (default)	1.00	1.00	8.99 (0.16)	–	–	8
purchase	RIPPER (increased pruning)	1.00	1.00	–	–	–	1
telecom	SVMExplainer	0.99	0.87	1.46 (0.56)	–	–	–
telecom	C4.5 (default)	0.98	0.98	66.79 (19.38)	331	661	–
telecom	C4.5 (increased pruning)	0.78	0.78	2.00 (0.00)	–	–	–
telecom	RIPPER (default)	0.99	0.99	199.11 (41.86)	–	–	208
telecom	RIPPER (increased pruning)	0.78	0.78	–	–	–	1
outsource	SVMExplainer	1.00	0.55	1.67 (1.08)	–	–	–
outsource	C4.5 (default)	1.00	1.00	13.53 (2.49)	42	83	–
outsource	C4.5 (increased pruning)	0.91	0.91	2.00 (0.00)	2	3	–
outsource	RIPPER (default)	1.00	1.00	23.26 (2.19)	–	–	25
outsource	RIPPER (increased pruning)	0.88	0.88	–	–	–	1
docflow	SVMExplainer	0.96	0.96	1.29 (0.62)	–	–	–
docflow	C4.5 (default)	0.98	0.98	22.50 (8.41)	18–	359	–
docflow	C4.5 (increased pruning)	0.82	0.82	6.95 (1.60)	11	21	–
docflow	RIPPER (default)	0.98	0.98	110.30 (27.60)	–	–	106
docflow	RIPPER (increased pruning)	0.73	0.73	4.88 (0.32)	–	–	3
incman	SVMExplainer	0.97	0.96	3.20 (1.21)	–	–	–
incman	C4.5 (default)	0.98	0.98	9.37 (2.53)	38	75	–
incman	C4.5 (increased pruning)	0.87	0.87	3.88 (0.33)	6	11	–
incman	RIPPER (default)	0.97	0.97	17.98 (4.24)	–	–	17
incman	RIPPER (increased pruning)	0.86	0.86	7.37 (1.11)	–	–	4

parameters, as detailed above. For the white box techniques, however, it is not possible to impose such limits on the model being built, so that we consider all correctly classified instances also as explainable.

Next, average and standard deviation for the length of instance explanations is given. For SECPI, the length of the best explanation corresponds to the number of attributes incorporated in the shortest (i.e. first) explanation found[1]. For C4.5 and RIPPER, however, explanations for specific instances have to be derived in some other manner when using these classifiers as white box techniques. As such, for each instance, we investigate which path through the rule base or decision tree was followed in order to return a class label and derive the length from this

[1]Consider for example the explanation "IF (*SometimesDirectlyFollows*$(a, b) = 0$ AND *SometimesDirectlyFollows*$(b, d) = 0$ AND *SometimesDirectlyFollows*$(f, g) = 0$) THEN Cluster$_i$" denoting that this instance would leave the cluster if the three attributes corresponding to the sometimes directly follows relations listed above would be set to zero. The length of this explanation is thus equal to 3.

Table 7.3 (continued): Results of the experimental evaluation comparing SECPI with C4.5 and RIPPER. This table shows the results for 10 clusters.

Event Log	Technique	10 Clusters					
		Acc.	Ex.I.	A.Ex.L. (StDev)	Nr.L.	T.S.	Nr.R.
incident	SVMExplainer	0.99	0.79	1.86 (1.19)	–	–	–
incident	C4.5 (default)	0.99	0.99	22.07 (5.20)	226	451	–
incident	C4.5 (increased pruning)	0.88	0.88	8.55 (1.22)	1–	19	–
incident	RIPPER (default)	1.00	1.00	86.90 (10.40)	–	–	121
incident	RIPPER (increased pruning)	0.85	0.85	6.79 (0.72)	–	–	4
purchase	SVMExplainer	1.00	1.00	4.11 (0.42)	–	–	–
purchase	C4.5 (default)	1.00	1.00	3.03 (0.35)	12	23	–
purchase	C4.5 (increased pruning)	1.00	1.00	2.00 (0.00)	–	–	–
purchase	RIPPER (default)	1.00	1.00	9.99 (0.21)	–	–	9
purchase	RIPPER (increased pruning)	1.00	1.00	–	–	–	1
telecom	SVMExplainer	0.99	0.88	1.56 (0.55)	–	–	–
telecom	C4.5 (default)	0.98	0.98	73.27 (20.95)	456	911	–
telecom	C4.5 (increased pruning)	0.80	0.80	17.10 (6.20)	27	53	–
telecom	RIPPER (default)	0.97	0.97	217.23 (40.11)	–	–	239
telecom	RIPPER (increased pruning)	0.78	0.78	–	–	–	1
outsource	SVMExplainer	1.00	0.55	1.62 (1.06)	–	–	–
outsource	C4.5 (default)	1.00	1.00	13.68 (2.30)	59	117	–
outsource	C4.5 (increased pruning)	0.94	0.94	3.92 (0.38)	4	7	–
outsource	RIPPER (default)	1.00	1.00	27.97 (3.14)	–	–	32
outsource	RIPPER (increased pruning)	0.94	0.94	2.00 (0.00)	–	–	3
docflow	SVMExplainer	0.96	0.96	1.41 (0.54)	–	–	–
docflow	C4.5 (default)	0.97	0.97	26.41 (10.61)	332	663	–
docflow	C4.5 (increased pruning)	0.81	0.81	8.15 (1.99)	19	37	–
docflow	RIPPER (default)	0.94	0.94	148.37 (29.89)	–	–	157
docflow	RIPPER (increased pruning)	0.70	0.70	12.76 (0.73)	–	–	5
incman	SVMExplainer	0.96	0.96	3.26 (1.31)	–	–	–
incman	C4.5 (default)	0.97	0.97	12.57 (3.64)	49	97	–
incman	C4.5 (increased pruning)	0.88	0.88	6.99 (1.86)	13	25	–
incman	RIPPER (default)	0.97	0.97	28.73 (5.06)	–	–	25
incman	RIPPER (increased pruning)	0.81	0.81	9.05 (1.86)	–	–	5

explanation (i.e. the number of attributes involved in the conjunction to reach this outcome). For C4.5 and RIPPER, we also report the size of the global model in terms of number of tree leaves, tree size, and number of rules.

The results show similar accuracy values for SECPI, C4.5 and RIPPER on all logs compared when using default parameter settings. Although not reported explicitly in this experiment, we note that SECPI generally comes with lower run time to construct the classifier following from the use of linear SVMs. However, we can observe that deriving explanations for trace instances directly from these white box classifiers leads to oversized explanations and multiple attribute checks, whereas SECPI is much better suited to find shorter, strong explanations. Also note that the white box classifiers construct explanations which potentially contain less-evident conjunctions, such as these based on an attribute (thus: behavior) being absent, rather than present. As was discussed before, it is much harder to determine how an instance would change if certain behavior would have been incorporated (as this behavior can present itself in many different ways in the trace) than whether certain behavior which currently is present would not have occurred. Furthermore, observe that the white box classifiers return large global models (seen for example by the number of rules returned by RIPPER). Increasing the amount of pruning performed indeed decreases the length of the retrieved explanations (and the size of the global model), but comes with two disadvantages. First, observe the drop in accuracy in cases where increased pruning is applied. Second, applying heightened pruning leads white box classifiers (especially RIPPER) to return a model containing a single rule: the majority prediction. As this rule is unusable in explaining the reasoning behind the clustering of instances, we cannot report the average explanation length in these cases (shown as "–" in Table 7.3). The overall conclusion of Table 7.3 is that SECPI is capable of providing much shorter and more useful (only behavior that is present in an instance is considered) explanations across event logs with different characteristics and for varying clustering sizes, with comparable accuracy to global white box classifiers.

7.6 Conclusion

In this chapter, SECPI (Search for Explanations of Clusters of Process Instances), a new technique for providing human understanding for trace clustering results was presented. The need for such a technique stems from the observation that typical trace clustering techniques do not provide sufficient insight into how a clustering solution is composed. Furthermore, it was argued that various potential alternative techniques, for instance a visual comparative analysis of the underlying discovered process models, fall short in resolving the problem at hand. Accordingly, our SECPI-algorithm aims to find a minimal set of domain-based characteristics such that, if these characteristics were not present, the instance would not remain in its current cluster. In this way, the technique is capable of discerning concise, individual rules that clearly explain why a certain instance is part of a cluster. In addition, its implementation as a ProM plugin allows for visualizing an explanation in the respective discovered process models underlying the clusters. In the experimental evaluation, SECPI was shown to strike a better balance between accuracy and comprehensibility as compared to typical white box classification techniques such as rule and decision tree learners. In future work, we foresee to expand on a number of closely related topics. First, we plan to inspect the impact of the attribute templates used as they play a crucial role in representing the (control-flow) domain. Also, we aim at investigating the incorporation of non-control-flow-based attributes. Second, aggregation of instance-level explanations is a worthwhile research track as well. The current implementation already supports the investigation of shared explanations amongst groups of instances, which is a preliminary approach to bring our explanation technique to the global level. However, we plan to investigate more intelligent rule clustering and visualization techniques for this purpose. In the future, we plan to make the global versus instance-based trade-off more configurable towards the end user. Finally, we will focus on practical use cases in which SECPI might prove beneficial. User-driven discovery of process model collections from event data is one such area where it can support the feedback mechanism. Furthermore, SECPI is also perfectly capable of relating exogenously defined clusters, e.g. high versus low cost instances, to process-specific control-flow characteristics, a feature often desired in business process improvement cycles.

Chapter 8

A Conformance Analysis Benchmarking Framework

"When you are content to be simply yourself
and don't compare or compete,
everyone will respect you."
– Lao Tzu

8.1 Introduction

This chapter introduces CoBeFra, a comprehensive, useable toolset for assessing the goodness of a process model or to easily benchmark the performance of different models against each other using multiple conformance checking metrics. The need for a comprehensive evaluation framework in the process mining domain was first articulated by Rozinat et al. [124]. As such we present the architecture of an extensible comprehensive benchmarking framework (abbreviated to CoBeFra, allowing for the consistent, comparative and repeatable calculation of process conformance metrics. Such an architecture is considered as being highly valuable for process mining researchers because it significantly facilitates the development and assessment of process discovery as well as conformance checking techniques.

In addition, we present a straightforward technique for event log generation from Petri nets for use within experimental setups, and present a benchmarking study which aims to uncover the relationship between process discovery techniques' performance and event log characteristics, which was performed using the two developed tools (CoBeFra and the event log generator).

8.2 Process Models Quality Metrics

A thorough overview of state of the art conformance checking techniques has already been provided in the introductory chapter—see Table 1.2, many of which have been included in our experimental setup of Chapter 3, which introduced two novel metrics towards assessing the precision and generalization capabilities of process models. Recall that the quality of a procedural process model (designed or discovered) can be judged along different perspectives. Figure 8.1 details that these perspectives can be categorized into two high-level dimensions: accuracy and comprehensibility. Each of these high-level quality dimensions can be further decomposed. For instance, the accuracy of a process model consists of the following three subdimensions: its recall (or: fitness), indicating the ability of the process model to replay or execute the observed behavior; its precision (or: appropriateness), denoting the model's ability to disallow (i.e. not support the execution of) unwanted, unseen behavior and thus its ability towards preventing underfitting the observed behavior; and last, the model's generalization capability, which indicates the model's ability to allow unseen but nevertheless desired or expected behavior and which can thus be seen as the counterpart of precision (i.e. avoiding overfitting). Next, from the viewpoint of comprehensibility, a distinction can be made between simplicity (or: complexity) and structuredness (or: understandability, entropy), the latter representing the ease of interpretation of the model while simplicity refers to the number of control-flow constructs present in the process model. As such, simplicity represents the principle that "all other things equal, a simpler explanation is better than a more complex one", a statement famously known as "Occam's razor". Oftentimes, simple counts of model elements are used as simplicity metrics.

8.2.1 Accuracy Metrics

As indicated above, the overall accuracy of a process model consists of three perspectives: recall, precision and generalization. Cook and Wolf [48] are to be considered the first researchers to quantify the relationship between process

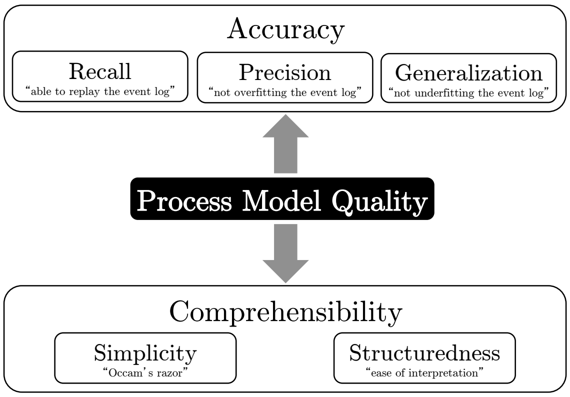


Figure 8.1: Quality dimensions for evaluating procedural process models .

models and event logs. Within the process mining domain, Rozinat et al. [100] defined the notions of fitness and appropriateness in a foundational study on conformance checking. Since, the domain has attracted the attention of many other researchers, as demonstrated by Table 1.2, which provides an overview of the most noteworthy conformance metrics and an indication of which metrics are currently already implemented in the proposed benchmarking framework.

In the remainder of this section, the metrics included in the CoBeFra framework are discussed in further detail.

8.2.1.1 Recall

Recall or fitness can be seen as the primordial accuracy evaluation perspective because it reflects how much behavior present in the event log is captured by the model, which can be regarded as an obvious quality requirement for designed or discovered models before continuing onwards with other analysis tasks. As such, many researchers have proposed metrics that quantify this dimension:

- *Fitness* (f) is a metric obtained by evaluating whether each trace in the event log can be reproduced by the generative model, in this case a Petri net. This procedure is called sequence replay [100, 217], as we have discussed in the first part of this dissertation. During replay, the transitions in the Petri net will produce and consume tokens to reflect the state transitions. Consequently, the fitness measure punishes for tokens that must be created additionally in the marked Petri net and also for tokens that remain after replay.

- *Proper Completion* (p_{PC}) is a coarse-grained metric which returns the percentage of traces without any missing or remaining tokens after trace replay [100]. Or, put differently, the percentage of traces for which a *Fitness* (f) value of 1 can be obtained.

- *Behavioral Recall* (r_B^p) as defined by Goedertier et al. [2] is the percentage of correctly classified positive events in the event log. Also by using sequence replay techniques, it is verified whether every positive event can be parsed by the model. Note that this metric differs from *Fitness* (f) since it denotes the ratio of (positive) events which could be replayed successfully by the process model, without incorporating the exact amount of missing or remaining tokens in the metric definition.

- *(Average) Alignment Based Trace Fitness* (f_a, f_a^{avg}) is a recent recall metric in the process mining domain [108–111]. In contrast to a large majority of recall metrics, this metric is based on aligning a process model and an event log, instead of replaying the log in the model. As such, the metric punishes for alignments which require model (log) moves without a corresponding move in the log (model). Other variants base the resulting value on the raw cost given to various kinds of misalignments rather than returning the percentages of non-aligned log-model moves.

- Weidlich et al. [102–104] have proposed a collection of *Behavioral Profile*-based conformance metrics. Instead of relying on state-based techniques that involve replaying the logs, the authors base their metrics on different types of constraints that a process model can impose for a pair of activities (i.e. the behavioral profile of the process model). We have included the Behavioral Profile Conformance metrics in the CoBeFra framework, both available in the form of the "standard" implementation provided by Weidlich et al. (using the jBPT library) and as an alternative implementation developed by the authors which is able to deal with event logs containing duplicate activities.

8.2.1.2 Precision

Precision entails that process models should prevent the execution of unseen and unwanted behavior (i.e. not underfitting the data). The following metrics are included in CoBeFra:

- *Advanced Behavioral Appropriateness* (a'_B) as defined in [100] is a footprint-based, rather coarse-grained precision metric, which in addition requires an exhaustive time consuming state space exploration.
- *Behavioral Specificity* (s^n_B) is the percentage of correctly classified negative events during sequence replay. As such, it is the counterpart of *Behavioral Recall* (r^p_B). Artificial negative events can be generated using the technique described in Chapter 3. However, the definition of the metric does not exclude the use of natural negative events or negative events stemming from other techniques. Note that, for many process model evaluation tasks, it is not the percentage of correctly classified negative events which is of real interest to the end user (the specificity), but rather the amount of "false positives", i.e. behavior which is allowed by the process model although this behavior is rejected in the given event log in the form of a negative event. This notion better corresponds with the true meaning of precision and is captured in the *Behavioral Precision* metric (p_B) as defined by De Weerdt et al. [107] and extended in this work in the form of a robust weighted variant (p^w_B), *Weighted Behavioral Precision.*
- *ETC Precision* (etc_P) is based on the construction of a prefix automaton for the event log at hand [105, 106]. By taking into account the number of so-called escaping edges while mapping the behavior in the event log with the behavior in the model, a state-of-the-art precision metric is defined by comparing the amount of "escaping" behavior the models allows in a given state compared to the log prefix automaton which is brought (using replay) in a comparable state. A very similar metric *Alignment Based Precision* (p_A) based on the concept of log-model alignments is described in [109].
- *One Align Precision* (a^1_p) and *Best Align Precision* (a_p) extend the etc_P metric by first aligning the log and the model [112]. In this way, the main problem of etc_P, i.e. the fact that precision is assessed as long as the trace under investigation can be replayed without error, is solved.

8.2.1.3 Generalization

Although models should be precise, generalizing beyond observed behavior is also a necessity. This is because assuming that all behavior is included in an event log is a far too strong completeness assumption. Metrics quantifying the generalization dimension should punish process models which are overly precise, thus not allowing unseen but very likely (not explicitly forbidden) behavior when taking into account the data in the event log:

- *Behavioral Generalization* (g_B) and *Weighted Behavioral Generalization* (g_B^w): as were introduced and elaborated in Chapter 3.
- *Alignment Based Probabilistic Generalization* (g_A) metric also starts from the principle of log-model alignment as described by Adriansyah et al. [109]. The authors propose a probabilistic (Bayesian) estimator which tries to assess the chance whether events will occur which exhibit behavior that was not seen before.

8.2.2 Comprehensibility Metrics

Because structuredness (ease of interpretation) is a difficult dimension to measure, many researchers focus on simplicity for quantifying the comprehensibility of process models. In [114], around twenty different metrics are defined to assess a model's compressibility. We have opted to include just a few count-based simplicity metrics by default in the CoBeFra framework: the number of arcs, nodes, places, transitions or cut vertices; the average node arc degree and the weighted place/transition node arc degree. Next to these metrics, the *Advanced Structural Appropriateness* (a_S') metric [100] is also included. This metric evaluates two specific design guidelines for expressing behavioral patterns, namely the occurrence of alternative duplicate tasks and redundant invisible tasks.

8.2.3 Combining Metrics

Another aspect that is becoming increasingly important in the conformance checking domain is the question on how to combine evaluation dimensions. In earlier work, we have proposed the use of the F-score to combine recall and precision metrics [107]. In [218], Buijs et al. propose to weight four evaluation dimensions, i.e. recall, precision, generalization and comprehensibility, in one metric for steering their genetic programming inspired Evolutionary Tree Miner (ETM) algorithm. As such, it was decided to add the F-score technique as well as a so-called “free weigher” to CoBeFra. The F-score allows to combine a fitness and precision metric with a configurable β value (F_1 being the harmonic mean of fitness and precision), whereas the free weigher allows to configure a linear combination of various metrics with configurable coefficients (weights).

8.3 Architectural Requirements

The following principles summarize the basic design requirements that were considered when developing the benchmarking framework in ProM.

8.3.1 Ease of Use

The first design requirement puts an emphasis on user friendliness. With this benchmarking framework, we aim to offer a straightforward interface for importing event logs and process models, for mapping each event’s class (i.e. its activity name and life cycle transition) to one of the activities (tasks, transitions) in the process model, for configuring the various metrics and, finally, for inspecting and exporting the obtained results. Furthermore, although the ProM framework allows for a clear differentiation between the end-user oriented graphical interface and internal logic, we have found that many plugins still rely on the presence of the user interface (UI), preventing an easy implementation in external (headless) scripts or tools. Therefore, in order to simplify experimental setups requiring a great deal of scripting and batch processing, we have strengthened the decoupling of the user interface and programming logic in CoBeFra, allowing each step (log and model setup, metric configuration and result processing) to be executed in a pure “headless” manner.

8.3.2 Reproducibility of Experiments

A second important design requirement consists of providing the functionality to reproduce results. By allowing to store input and metric configurations, it is very straightforward to repeat experiments over time.

8.3.3 Comparative Consistency

Consistency is safeguarded by a number of elements. First of all, because the same initially configured model-log mapping is used across all metrics, no obvious mistakes are made to this regard. Furthermore, by streamlining the metric configuration step, users will have less trouble in configuring the often huge amount of configurations across different experiments, which promotes consistency of results.

8.3.4 Computation Management

A limitation of the ProM framework is that it is not straightforward to set up an environment in which multiple conformance checks can be executed at the same time. Therefore, our approach is to first allow the user to configure all model-log inputs together with the list of desired metrics to run and their configuration. Afterwards, the calculation of the metrics itself is started; we have implemented a computation manager which allows to run multiple metric calculations in parallel. The next section provides more technical details. Finally, since some metrics can consume a large amount of time before finishing, an option to both manually and automatically cancel a metric's calculation procedure was added as well. This allows researchers to easily impose time-based bounds while running experiments.

8.3.5 Extensibility

Finally, the framework is designed so to be easily extended with other or future conformance metrics. A large number of metrics have already been implemented, but we invite scholars and authors to implement their work in CoBeFra as well. Making a ProM conformance checking plugin available in CoBeFra is quite straightforward, especially if best practises were followed during the development of the original ProM plugin, since CoBeFra also uses and allows to easily tie in with the ProM architecture.

8.4 Framework Architecture

This section describes the technical architecture of the framework in more detail. Particular attention is paid to the topics of model-log mapping, separation between domain logic and user interface, legacy support and parallelism.

8.4.1 General Architecture

The CoBeFra framework is integrated in ProM 6 and reuses various existing libraries and components available in ProM. Figure 8.2 provides a schematic overview of the developed architecture. At the root of the architecture is a global application controller which is responsible for managing the general user interface, the flow between the initial input setup, metric configuration and result handling, and also provides an application programming interface (API) to perform all steps programmatically. CoBeFra can be started as any other ProM plugin, and allows to optionally specify a previously saved project to resume or restart an experiment. Importing and exporting of input objects is done through the standard ProM provided architecture as well, and visualization of obtained metric results was decoupled into a separate visualization plugin. Once CoBeFra is started, the framework discards all ProM-specific dependencies so that it can easily be run as a stand-alone application as well.

8.4.2 Particular Items

8.4.2.1 Mapping Process Models with Event Logs

Although it appears straightforward to link process model activities (Petri net transitions) with events in a process model after executing a process discovery algorithm or during model execution, this link or “mapping” is lost once both objects are saved separately, or are modified asynchronously. Mapping a model to a log is thus a crucial step in each conformance checking analysis task, as there needs to be a clear and unambiguously defined relation between process model activity elements and log events (see the μ function in the introductory preliminaries).

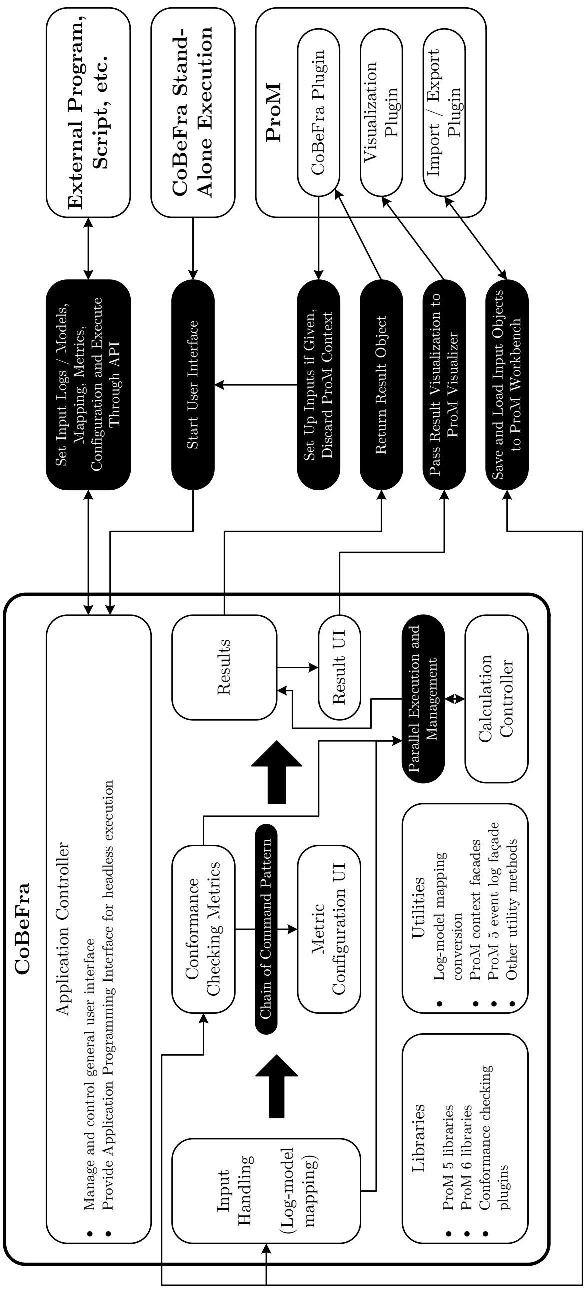

External Program, Script, etc.
CoBeFra Stand-Alone Execution
ProM
CoBeFra Plugin
Visualization Plugin
Import / Export Plugin
Set Input Logs / Models, Mapping, Metrics, Configuration and Execute Through API
Start User Interface
Set Up Inputs if Given, Discard ProM Context
Return Result Object
Pass Result Visualization to ProM Visualizer
Save and Load Input Objects to ProM Workbench
CoBeFra
Application Controller
Manage and control general user interface
Provide Application Programming Interface for headless execution
Results
Result UI
Parallel Execution and Management
Calculation Controller
Conformance Checking Metrics
Chain of Command Pattern
Metric Configuration UI
Utilities
Log-model mapping conversion
ProM context facades
ProM 5 event log façade
Other utility methods
Input Handling
(Log-model mapping)
Libraries
ProM 5 libraries
ProM 6 libraries
Conformance checking plugins

Figure 8.2: Schematic overview of the CoBeFra architecture.

We have found that many conformance checking plugins in ProM provide alternative implementations of the above described mappings; to be exact, six different methods to describe the link between event log activities and Petri net transitions were discovered. Furthermore, many of them do not deal with the possibility of "blocked" transitions, and have different ways to specify "invisible" transitions. Finally, many existing mappers come with UI components that allow users to construct a mapping, but many of them are cumbersome to use, especially when having to deal with large models containing many transitions or event classes, or when having to execute repeated experiments. Since even a single incorrectly mapped transition may lead to largely skewed quality results, the mapping process should try to prevent errors as best as possible. Therefore, we have implemented an unified mapping system to deal with the above issues. A mapping must be performed by the user between each model-event log pair, but the provided user interface allows to configure mappings for multiple models towards the same log file simultaneously. Furthermore, an intelligent string matching routine pre-maps transitions to event classes whenever possible, providing an option to the user to immediately map the non-automatically matched transitions as being invisible or blocked. Finally, autocompletion-enabled UI controls allow users to rapidly assign the remaining transitions to an event class. Our mapping object serves as the basis to save and load model-event log combinations, preventing having to re-enter the mapping between an event log and process model every time an experiment is ran. Finally, our collection of utility classes provides support to convert our mapping to the different other mapping representations used by various metrics in a transparent manner, so that the step of constructing a mapping is completely decoupled from the configuration and execution of the conformance checking metrics.

8.4.2.2 User Interface and Program Logic Separation

A second architectural aspect we wish to emphasize is the great deal of attention given to separating user interface components with the actual domain model and program logic (e.g. calculation of metric values). To do so, the conformance checking metrics themselves have been implemented as classes which fulfill their metric contract by implementing a "Metric" interface. Precisely speaking, this contract imposes the constraints on an implementing metric that it must be able to return a numeric result value when asked to, and that the implemented metric must be configurable via a standard means, that is by getting and setting key-value attributes. The latter also allows for easy serialization of a metric configuration, which is used to save and load experimental setups. These metric calculating classes are completely UI-agnostic. Next, to provide an easy means to configure each metric, UI components can be defined which take a metric class as an input and allow to modify a metric's configuration via a visual interface. The UI components are annotated (using standard Java Annotations) with information to specify which metric classes can be configured through their provided interface. When starting CoBeFra, an internal repository of all UI providing classes is constructed. When a user wishes to configure a specific metric, all matching UI classes are iterated through (using the chain of command programming pattern) and shown. Users accessing CoBeFra using the API can completely bypass the user interface and directly configure the metrics themselves using the key-value store discussed above. Finally, it should be noted that some existing ProM plugins rely heavily on (or assume) the presence of a user interface, for example by hooking logical segments directly into UI actions (such as pressing a button). To cleanly implement these metrics in CoBeFra, some code was copied and refactored to remove the UI-relying segments. Another issue often encountered when inspecting ProM plugins is that authors assume the presence of the ProM-provided "plugin context" object (for example to report progress back to the ProM framework), but are not able to continue when this context is absent or – worse – when this context is not specifically defined as being a UI-enabled (ProM allows to specify a command line interface context, but some plugins specifically check for the UI-based context, even when this is not required). Therefore, we have also defined a utility class which wraps around the ProM context object definition by overloading UI-specific functions, which can be passed directly to a conformance checking metric's methods.

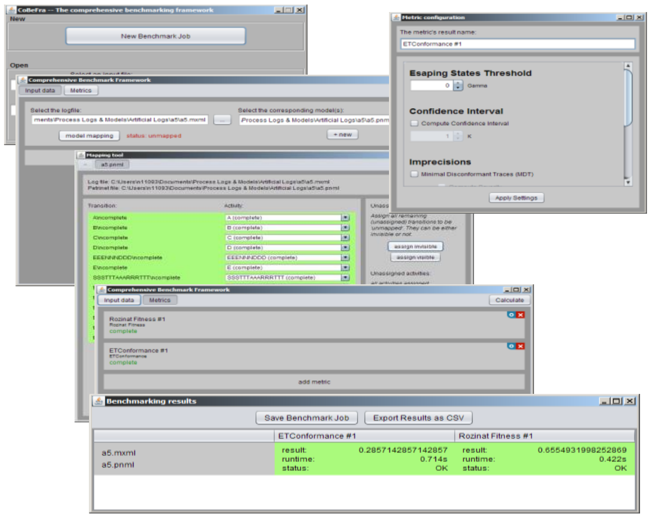
8.4.2.3 Support for ProM 5 Metrics

An important contribution of our conformance checking framework is that it includes some well-known conformance checking metrics which are only implemented in ProM 5, namely Fitness (f), Advanced Behavioral Appropriateness (a'_B) and Advanced Structural Appropriateness (a'_S). The ProM 5 architecture greatly differs from ProM 6. Not only is the notion of plugin contexts missing (instead, progress listener objects are used), but the internal way event logs are represented is dissimilar as well (ProM 5 does not use the XES standard). Since it would be infeasible to redundantly rewrite large parts of ProM 5 conformance checking metrics (possibly leading to new errors as well), we have defined facade classes wrapped around ProM 5's progress and event log related definitions. Not only does this allow to use ProM 5-only conformance checking metrics, but also to directly load in XES-files and use them in combination with these metrics, which is not possible when using ProM 5 itself (which only accepts the MXML-format). Since the facade objects overload the event log reading methods on the fly, this process is transparent to the end-user.

8.4.2.4 Computation Management and Parallelism

The CoBeFra benchmarking framework allows to run multiple metric calculations in parallel, thus speeding up experiments, especially on multi-core systems. Next to parallelization, the ability to (automatically) cancel a metric's calculation routine is also an important requirement, as this allows users to impose time constraints on the executed experiments. To implement these features, we have opted towards using a multi-process architecture (rather than multi-threaded). This approach entails some useful advantages. Not only does this avoid having to wrap each metric's calculation routine in a threaded worker unit (possibly leading to synchronization issues or other multi-threading related problems), but also prevents that a fatal error in one metric would halt the complete experiment and discard its results. Management of processes is a simple matter. Users can specify a time limit after which the metric's process is killed. To communicate back and forth from the CoBeFra host process and the metric executing processes, we make use of standard interprocess communication practices by redirecting standard POSIX input, output and error streams. The CoBeFra host process first sends the metric's configuration, event log and model to the process using a compressed stream, after which the host process waits until a result is sent back on the metric's process output stream, or an error is thrown on the metric's process error stream, which is captured and shown to the end user in the final result overview as well. Although currently unimplemented, it can indeed be remarked that this setup theoretically allows to parallelize conformance checking not only on the same host machine, but also across multiple machines over a networked architecture. Exploring the possibilities towards enabling conformance checking to be executed on a computing grid is acknowledged to be an interesting path for further work. Other possibilities towards future work and current limitations are listed in the following section.

This discussion on architecture concludes the presentation of our comprehensive benchmarking framework (CoBeFra). Figure 8.3 depicts a screen capture of the benchmarking tool. We invite peers and researchers studying conformance checking to contribute and improve upon the framework.



Select the logfile: ments\Process Logs & Models\Artificial Logs\a5\a5.mxml

Select the corresponding model(s): Process Logs & Models\Artificial Logs\a5\a5.pnml

status: unmapped

Log file: C:\Users\n11093\Documents\Process Logs & Models\Artificial Logs\a5\a5.mxml
Petrinet file: C:\Users\n11093\Documents\Process Logs & Models\Artificial Logs\a5\a5.pnml

Transition:	Activity:
A\ncomplete	A (complete)
B\ncomplete	B (complete)
C\ncomplete	C (complete)
D\ncomplete	D (complete)
EEENNNDDD\ncomplete	EEENNNDDD (complete)
E\ncomplete	E (complete)
SSSTTTAAARRRTTT\ncomplete	SSSTTTAAARRRTTT (complete)

Assign all remaining (unassigned) transitions to be 'unmapped'. They can be either invisible or not.

The metric's result name: ETConformance #1

Esaping States Threshold: 0 Gamma

Confidence Interval: Compute Confidence Interval; 1 K

Imprecisions: Minimal Disconformant Traces (MDT)

Rozinat Fitness #1 — Rozinat Fitness — complete

ETConformance #1 — ETConformance — complete

	ETConformance #1	Rozinat Fitness #1
a5.mxml a5.pnml	result: 0.2857142857142857 runtime: 0.714s status: OK	result: 0.6554931998252869 runtime: 0.422s status: OK

Figure 8.3: Screen captures of the CoBeFra user interface.

8.5 Future Work

Although the proposed CoBeFra framework satisfies the desired design requirements and already includes a great deal of conformance checking metrics, different potential improvements exist, summarized in the following overview.

8.5.1 Other Process Model Representations

The currently-available implementation of CoBeFra only includes Petri net oriented metrics. However, many other process representations exist for which an extensive conformance checking framework would be beneficial. For instance, de Leoni et al. [219] apply the principle of model-log alignment on declarative process models (Declare models). Accordingly, we plan to investigate how other representations can be incorporated in CoBeFra.

8.5.2 Graphical Output

The current version of CoBeFra presents the metric results as a table, exportable as a CSV-file. Graphical outputs such as charts, graphs, pareto-maps, etc. will make the resulting output more user friendly. This will also be explored in future work.

8.5.3 Root Cause Analysis

It is important to note that while conformance checking metric values give a good initial indication regarding the quality of a process model (or the level of conformance of an event log), it is also important to know where errors occurred in the process model (or perhaps time frame in the event log). Some conformance checking metrics already allow to do so. Standardizing this feature over multiple metrics is a challenging task which is perhaps out of scope for a benchmarking oriented framework.

8.5.4 Automatizing Process Discovery

CoBeFra allows to automatize a great deal of experimental setup, configuration and management tasks when executing a quality assessment study. Nevertheless, the presence of process models and event logs is assumed, which may require that users first run several process discovery algorithms to obtain the set of desired process models to be checked. Automatizing the task of process discovery (perhaps in a separate tool) is a possible path for future consideration, but was not included in the current scope of the project.

8.5.5 Fine Tuning Event Classification and Non-control-flow Conformance Checking

The benchmarking framework proposed in this chapter mainly focused on the control-flow perspective of process models. It is possible to incorporate other perspectives (e.g. data or resource-based views) in the task of conformance checking as well. To do so, the current mapping system has to be modified to allow for more configurability regarding the classification of events. In addition, conformance checking metrics have to be incorporated which are able to check on these other dimensions. We note, however, that much of the conformance checking literature so far has focused on the control-flow perspective.

8.5.6 Standard Validation Event Log Set

Especially for researchers developing a new process discovery algorithm, or conformance checking technique, it is necessary to compare ones own work with the efforts of other peers. The question then becomes which input data set(s) (event logs) should be used to do so. Ideally, a "standard" data set should be constructed, including logs of different sizes (trace variant count, trace instances count, activity type count, etc.) and of different known quality level (e.g. flower model). Additionally, a standard data set should also include some robustness checks, for example by including very noisy (purely random) event logs. Constructing such an ideal event log set is an interesting challenge which would be a fitting complementary item for the proposed benchmarking framework.

8.5.7 Fine Tuning Computation Management

Although the current version of CoBeFra allows to parallelize the calculation of conformance checking metrics, it should be noted that executing too many tasks at once may bias run time results of the metric. The current recommendation is thus to keep the number of parallel executions below the amount of available processing cores (the host operating system will then attempt to distribute each metric as best as possible over the possible cores), but it is also possible to further fine tune the computational management by strictly binding a process to a specific processing core and only allowing one metric execution per core. Furthermore, we have indicated that an interesting possibility for future work is to leverage the multi-process execution architecture so that conformance checking metrics can be executed on a computing grid over a network. Finally, while the current computation manager allows to set a time bound after which a metric is automatically canceled, a second option could be added which sets a bound on the amount of memory which may be consumed by a conformance checking metric.

8.5.8 Cross-Validation

Cross-validation-based evaluation is not frequently applied in the area of process mining, since process mining tasks are typically applied for descriptive, rather than predictive analysis purposes. Furthermore, sensibly splitting a given event log is not easy. Still, it should be possible to implement cross-validation (or train/test splitting, jackknifing) methods in a conformance checking setting by making guided decisions (instead of choosing randomly) when splitting the event log.

8.5.9 Recommending Process Mining Techniques

With the large amount of process discovery techniques being proposed, there has been a rising interest in the construction of recommendation engines to propose suitable discovery techniques given the characteristics of the input event log and the preferences of the end user [220–222].

Against this background, CoBeFra can be utilized as a suitable calculation engine to construct a repository of performance metrics for discovery technique–event log–conformance metric combinations. CoBeFra was recently applied as such in [223].

8.6 Petri Net Based Event Log Generation

In this section, we introduce a ProM plugin which allows for straightforward event log generation, using token-based simulation driven by Petri net models. Although a large number of tools already exist for the simulation and analysis of Petri nets (CPN Tools being among the most notable), no technique exists which allows for the rapid generation of event logs (a collection of execution traces) based on a user-supplied Petri net in a straightforward manner. Our proposed tool focuses on ease of use, provides different simulation strategies and configuration options covering most standard use cases, and outputs the desired event log in a format which can immediately be utilized in subsequent steps within a process mining analysis workflow.

This plugin was used to enable the construction of synthetic event logs in the experiments included in this dissertation.

8.6.1 Rationale

Although event logs are regarded to be the focal point of analysis within the field of process mining, no straightforward approach or tool exists to construct these data repositories in a synthetic manner. However, such technique could be useful in many academic and other use cases. To generate synthetic event logs, scholars and practitioners have, so far, mainly resorted to using the following tools and approaches:

- Firstly, CPN Tools [224], which can be regarded as, by far, the most complete and advanced suite for coloured Petri net modeling, simulation and analysis, but comes with a steep learning curve and is hard to use in the use case of modeling a (non-coulored) Petri net and deriving a collection of simulated traces.
- Second, the Process Log Generator [159], which randomly generates models based on user-supplied criteria and can also provide a related simulated event log, but does not provide means to modify the generated model or change simulation options.
- The final approach consists of direct construction of event logs without any formal semantic model driving a simulation, other than perhaps some stochastic statistical process (e.g. randomly constructing traces from a pool of activity labels). This last approach has the drawback that the constructed synthetic event logs are of less use, as process mining techniques assume that there is some underlying process model driving the execution of activities.

To help remedy this gap within the current offering of tools, we present a ProM plugin which allows for straightforward event log generation using token-based simulation driven by Petri net models. Our proposed tool focuses on ease of use, provides different simulation strategies and configuration options covering most standard use cases, and outputs the generated event logs in a native and standardized XES format[1], which can be immediately utilized in subsequent steps within a process mining analysis workflow.

8.6.2 Objectives

This subsection outlines a description of the purpose and applicative domain of the tool.

[1]Extensible Event Stream, see: `http://www.xes-standard.org/xesstandarddefinition`.

In a nutshell, the main objective of our developed tool was to offer an easy way to generate an event log from a given Petri net model which also enables user to configure some general options relevant to process mining practitioners and researchers, and outputs an event log which is directly useable in ProM without having to perform any intermediary steps. Concerning the supplied Petri net, we have deliberately chosen to regard the modeling of such models as being out of scope for our plugin, as there exist many excellent tools already which allow for the rapid modeling of such models, which can easily be imported in ProM. However, the simulation of an event log in these tools is seldom supported in a user-friendly manner.

The development of this plugin has followed from a real "need" experienced when setting up experiments to compare the performance of process mining discovery and other algorithms, which—in most scenarios—are composed out of both real-life and synthetic event logs. We thus can enumerate our objectives as follows:

- *Integration in ProM* The plugin should be integrated into ProM, rather than a stand-alone package, as the ProM framework is heavily utilized by process mining researchers and practitioners.
- *Relevant Configuration Options* The plugin does not allow to modify supplied Petri net models, as there exists many tools already supporting the creation (and validation) of such models. However, the plugin should allow to configure relevant options, such as the visibility of transitions.
- *Native Event Log Format Generated* event logs should be created as XES files, the event log format native to ProM. Many tools already allow to generate event logs as key-value pairs, comma separated value (CSV) files or some proprietary format, which requires an additional ETL step to import the data into ProM, which we want to avoid.
- *Ease of Use* The plugin should be straightforward to use with sensible defaults.

8.6.3 Functionality

Our plugin is started from within ProM and expects only a Petri net object as an input (which can easily be imported in ProM from many available file formats, such as PNML).

Figure 8.4 guides the reader through the various steps of the plugin. After ProM is opened and a Petri net model has been imported, this Petri net is used as an input object for the plugin, which can then be invoked. Figure 8.4(a) shows the first configuration panel, and allows one to configure the following general simulation options:

- *Simulation Method* Available methods are: random generation, complete generation or grouped path (distinct) generation. The first method randomly executes enabled transitions (configurable with weights, see below) in the Petri net until an end condition is reached. The second method performs a complete exploration of the Petri net state graph to generate all possible traces (bounded by the "Maximum times marking seen" option, see below), while the final method ensures the generation of distinct traces, i.e. the same event log trace cannot be utilized twice.
- *Number of Generated Traces* Amount of traces to generate (not available for complete generation).
- *Mininum/Maximum Traces to Add for each Generated Sequence* How many "copies" of each generated trace should be added to the event log. The number is uniformly chosen between the minimum and maximum. Default is one (1) for minimum and maximum values, so the event log ends up containing exactly the amount of traces as configured by the "Number of generated traces"-option (for random and grouped path generation).
- *Maximum Times Marking Seen* How many times the same marking may be seen within a trace. This option bounds the amount of times a loop will be followed in the process model.
- *Only Include Traces that Reach End State* The simulation of a trace ends whenever there is no enabled transition which can be executed anymore (enabled traces might be available but prohibited by the "Maximum times marking seen"-option). In this case, the sequence of transitions is converted to a sequence of trace events and added to the event log. When the "Only include traces that reach end state"-option is set, traces are only added when there is a token in a sink place (a place not containing outgoing arcs). Naturally, when the users supplies a workflow net, only one such sink place is present.
- *Only Include Traces Without Remaining Tokens* Same as above, but now, traces are only added when *all* the tokens in the final marking are in a

sink place. The default option enables the latter two options, which serves as a sensible default in most use cases to generate "valid", expected traces.

The following configuration screen, shown in Figure 8.4(b), allows to configure global trace timing options. This includes:

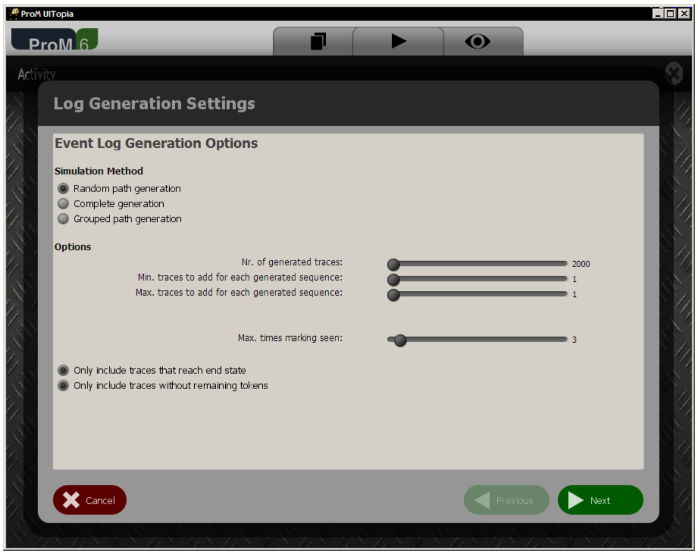
- *Anchor Point* A date/time stamp used as a point of reference in following options.
- *Trace Movement* Both "moving" and "fixed" traces are available. For fixed traces, the initial start time for each trace is set equal to the given anchor point. For moving traces, the starting point of the first trace is set equal to the anchor point, but the starting point of each following trace is set to the end (complete time of final activity) of the previous trace.
- *Variance* Average and standard deviation in seconds to add variance to the starting point of each trace. This allows e.g. for partial overlap or volatile, randomized starting points.

The next configuration screen in the wizard, depicted in Figure 8.4(c), allows to map each Petri net transition to an activity label in the event log to be generated. Transitions can be set to invisible by leaving the label blank, and multiple transitions can be mapped to the same label. The reason why we allow users to do this is because not all Petri net modeling tools support saving Petri nets with an explicit distinction between invisible and visible transitions, and so that users can assign separate, understandable labels for duplicate transitions in a modeler (e.g. "activityDup_1" and "activityDup_2") while still mapping them to the same event log activity ("activityDup").

In addition, this screen also allows one to set transition "weights", which allow to drive the simulation towards choosing one enabled transition over others when multiple choices are available. The chance to select a transition $t \in T$ (all Petri net transitions) at each step is thus:

$$\begin{cases} 0 & \mathit{if} \neg \mathit{enabled}(t) \\ \frac{\mathit{weight}(t)}{\sum_{t \in T:\mathit{enabled}(t)} \mathit{weight}(t)} & \mathit{if}\, \mathit{enabled}(t) \end{cases}$$

Finally, for each transition, the user can also display a "timings"-window (see Figure 8.4(c)) which allows one to set averages and standard deviations for the lead times (duration) and idle times (waiting time before an activity start) of activities. The standard option inserts a fixed idle time of one minute between each activity and only generates a "complete" event for each activity (atomic activities), which is sufficient to generate event logs which can be used by the multitude of miners and other analysis techniques. Finally, for each transition, the user can also display a "resources"-window (see Figure 8.4(d)) which allows one to configure the possible resources executing this activity, together with weightings. This is beneficial for users wanting to generate event logs to be used in social analysis scenarios.



Log Generation Settings

Event Log Generation Options

Simulation Method

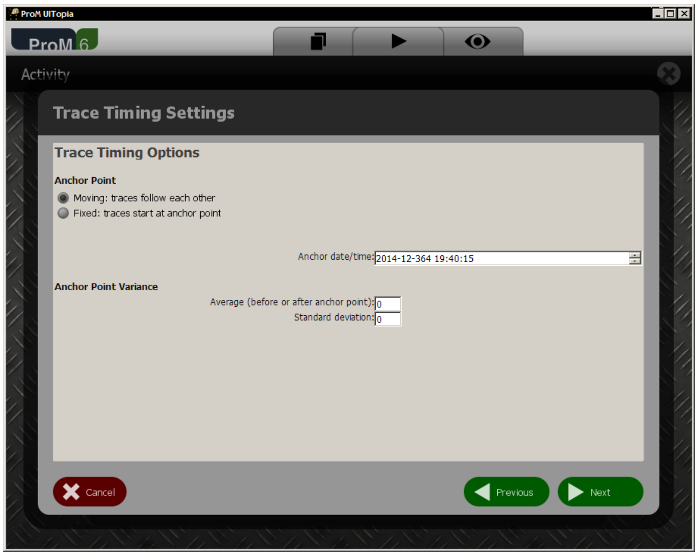
- [✓ Random path generation]
- Complete generation
- Grouped path generation

Options

Nr. of generated traces:	2000
Min. traces to add for each generated sequence:	1
Max. traces to add for each generated sequence:	1
Max. times marking seen:	3

- [✓ Only include traces that reach end state]
- [✓ Only include traces without remaining tokens]

(a) Configuring simulation options.



Trace Timing Settings

Trace Timing Options

Anchor Point

- [✓ Moving: traces follow each other]
- Fixed: traces start at anchor point

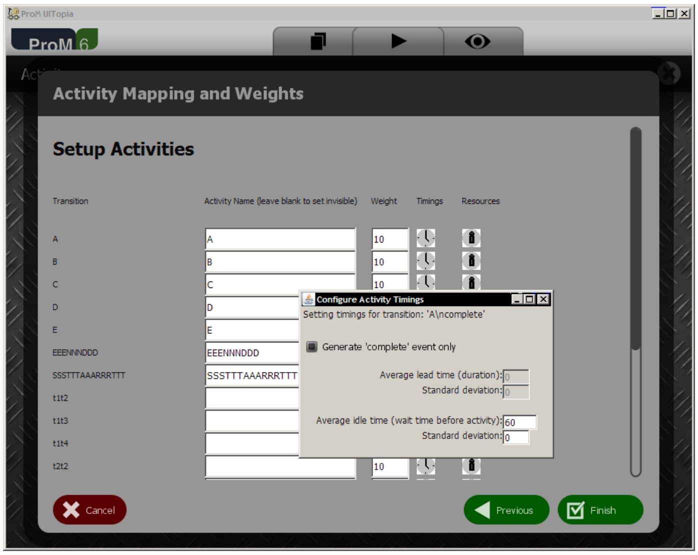
Anchor date/time: 2014-12-364 19:40:15

Anchor Point Variance

Average (before or after anchor point): 0
Standard deviation: 0

(b) Configuring global trace timings.

Figure 8.4: Collection of screen shots depicting the various steps of the developed plugin.



Activity Mapping and Weights

Setup Activities

Transition	Activity Name (leave blank to set invisible)	Weight	Timings	Resources
A	A	10		
B	B	10		
C	C	10		
D	D			
E	E			
EEENNNDDD	EEENNNDDD			
SSSTTTAAARRRTTT	SSSTTTAAARRRTTT			
t1t2				
t1t3				
t1t4				
t2t2		10		

Configure Activity Timings

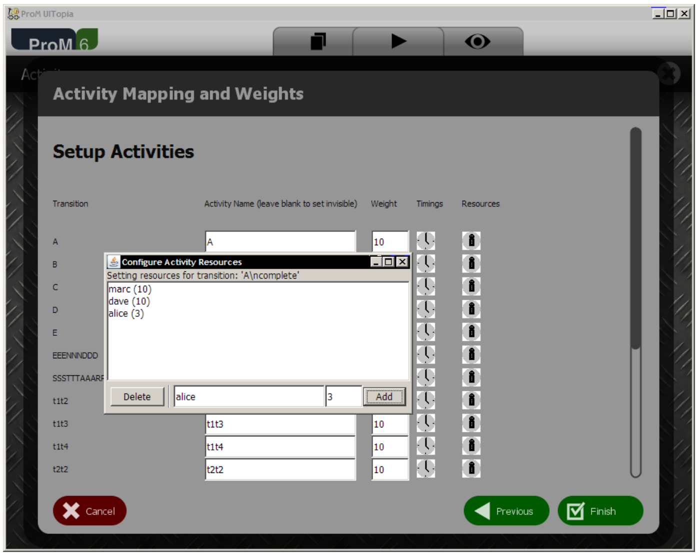
Setting timings for transition: 'A\ncomplete'

Generate 'complete' event only

Average lead time (duration): 0
Standard deviation: 0

Average idle time (wait time before activity): 60
Standard deviation: 0

(c) Configuring activities and activity timings.



Activity Mapping and Weights

Setup Activities

Transition	Activity Name (leave blank to set invisible)	Weight	Timings	Resources
A	A	10		
B				
C				
D				
E				
EEENNNDDD				
SSSTTTAAARF				
t1t2				
t1t3	t1t3	10		
t1t4	t1t4	10		
t2t2	t2t2	10		

Configure Activity Resources

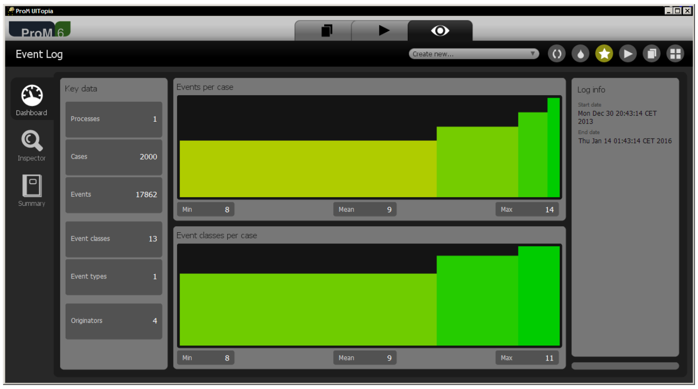
Setting resources for transition: 'A\ncomplete'

marc (10)
dave (10)
alice (3)

Delete | alice | 3 | Add

(d) Configuring activity resources.

Figure 8.4 (continued): Collection of screen shots depicting the various steps of the developed plugin.



Event Log

Key data

Processes	1
Cases	2000
Events	17862
Event classes	13
Event types	1
Originators	4

Events per case

Min 8 Mean 9 Max 14

Event classes per case

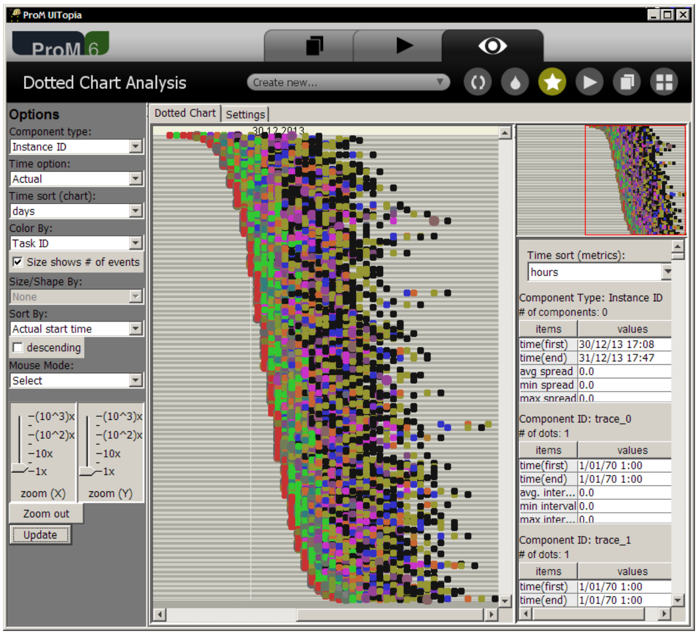
Min 8 Mean 9 Max 11

Log info

Start date
Mon Dec 30 20:43:14 CET 2013

End date
Thu Jan 14 01:43:14 CET 2016

(e) Generated event log opens in ProM



Dotted Chart Analysis

Options

Component type: Instance ID

Time option: Actual

Time sort (chart): days

Color By: Task ID

☑ Size shows # of events

Size/Shape By: None

Sort By: Actual start time

☐ descending

Mouse Mode: Select

zoom (X): -(10^3)x -(10^2)x -10x -1x

zoom (Y): -(10^3)x -(10^2)x -10x -1x

Time sort (metrics): hours

Component Type: Instance ID
# of components: 0

items	values
time(first)	30/12/13 17:08
time(end)	31/12/13 17:47
avg spread	0.0
min spread	0.0
max spread	0.0

Component ID: trace_0
# of dots: 1

items	values
time(first)	1/01/70 1:00
time(end)	1/01/70 1:00
avg. inter...	0.0
min interval	0.0
max inter...	0.0

Component ID: trace_1
# of dots: 1

items	values
time(first)	1/01/70 1:00
time(end)	1/01/70 1:00

(f) Analyzing the generated event log using dotted charts.

Figure 8.4 (continued): Collection of screen shots depicting the various steps of the developed plugin.

After finishing the configuration, the generation process is initialized. Once completed, the generated event log is opened in ProM (see Figure 8.4(e)), where it can be further analyzed or exported to disk, using not only the XES file format but any format for which a ProM exporter is available. Figure 8.4(f) shows a dotted chart analysis of the generated event log, illustrating at the same time the time variance induced in the generated log.

8.6.4 Architecture and Use Cases

Our tool is build on top of the ProM framework and thus re-uses components of the latter. In particular, handling of Petri net models and event log concepts is based on ProM's internal models, as to maximize interoperability with existing plugins. Configuration screens are shown using ProM's default widgets.

To simulate traces from Petri nets, we make use of internally developed components, rather than relying on external libraries. Our plugin does not utilize any native components, and thus runs on every platform where Java and ProM are available. Data interchange with other tools is provided using ProM's standard import/export mechanisms. That is, Petri nets are loaded in using ProM's available file readers, and generated event logs are opened natively in ProM until the user is ready to export them. This also allows users to easily apply post-generation steps such as event log noise generation by means of other plugins.

We believe our tool to be useful to scholars, practitioners and students working in the field of process mining. The authors have already utilized the presented plugin, both in research and educational settings. Some common use cases benefiting from generated synthetic logs include:

- Performing empirical experiments using a controlled data set.
- Constructing example data sets to use for educational purposes.
- Constructing illustrative data sets to present to usefulness of analysis techniques within a business context, where real-life data is not immediately available (due to technical or other reasons, e.g. privacy concerns).

8.6.5 Comparison

This section briefly compares our developed plugin against other tools and techniques. We hereby keep in mind the core objectives of our tool, offering a straightforward method to generate event logs from a Petri net in an environment familiar to process miners.

8.6.5.1 CPN Tools

CPN Tools is the most widespread and advanced tool for editing, simulating and analyzing coloured Petri nets [224, 225][2]. Although the latest versions have made it easier to model a simple (non-coloured) Petri net, the particular user interface of this tool comes with a rather steep learning curve. In addition, performing a repeated simulation run which can be outputted to an event log requires some setting up and extract-load-transform steps using ProMimport [33, 226]. Plugins have also been made available in ProM which allow for more straightforward generation of event logs from CPN models in ProM [227], though knowledge of CPN modeling is still assumed (furthermore, in ProM 6.3 and CPN Tools 4.0—the latest stable releases of both—the simulation plugins crash due to not being able to handle the "real" data type of CPN models). By our plugin, we try to offer a simple alternative to CPN Tools when the ultimate goal is the generation of an event log given a Petri net.

8.6.5.2 Process Log Generator

The Process Log Generator (PLG) is developed by Burattin et al. [159], and allows users to generate random business processes by only specifying some simple parameters. The tool allows one to generate an event log from the generated process model, but does not allow to make modifications to the randomized process model or load in a Petri net designed in another modeler.

8.6.5.3 General Petri net Modeling Tools

Including ARIS, FileNet Designer, FLOWer, PIPE, Protos, Renew, WoPeD, Yasper and other tools mentioned by the Petri Nets Tool Database [228]. A plethora of Petri net oriented modeling tools exist, with PIPE [229], Renew [230], WoPeD [231, 232] and Yasper [233] being free, open and favored offerings by the process mining community. Many of these tools also allow to perform Petri net analysis, soundness verification, and to play the Petri net "token game". However, although many of these tools fully support the semantics of Petri net-based execution, they lack support to generate an event log as a whole using some configurable parameters. The purpose of our plugin is not to replace these tools,

[2]See `http://cpntools.org`.

as they are still to be used to model (and verify) the Petri net from which an event log should be generated.

8.6.5.4 Other (Petri net oriented) Simulation Tools

Including Arena, FileNet Simulator, GPenSIM, HiPS, Petri .NET Simulator, Stateflow, TINA. Other general purpose tools focus less on the modeling of Petri nets, but more on the simulation perspective. However, by “simulation”, the statistical analysis of Petri net models is meant, rather than simulating (and saving) execution traces. Users are able to retrieve far reaching reports concerning utilization rates and mean durations, but no historic collection of traces can be saved. Furthermore, many of these tools are advanced and come with a steep learning curve.

8.6.5.5 Spreadsheets and Other Custom Ad-hoc Approaches

Users also resort to using spreadsheets (with macros and formulas) or programming scripts against the OpenXES API to generate synthetic, randomized event logs. However, the usability of such techniques is less proven, as there now is no “true” process model driving the generation of traces. Furthermore, as spreadsheet-based approaches lack the capability to output a native event log, they also have to be converted to another format in a post-step.

A drawback of our proposed tool is that it currently only support event log generation from Petri net models, but given this representation’s popularity within the area of process mining and the availability of conversion plugins which allow to convert many other representational forms to a Petri net, this is not a limiting problem in practice. In future version of the tool, however, we plan to incorporate ways to handle more advanced constructs, such as inhibitor and reset arcs, in Petri nets. Other plans for future work incorporate: improving the look and feel of the user interface, allowing to save and load simulation settings, selecting custom distributions for random sampling of transitions, resources and timings.

8.7 A Benchmarking Study

This section presents a benchmarking study which was performed using the tools presented above with the objective to uncover the relationship between event log characteristics and process discovery techniques. Given the large amount of process discovery algorithms being available, practitioners wanting to use process mining in real-life applications and environments are confronted with a real issue: having so many techniques available, it becomes difficult to choose the most appropriate one for a given situation without possessing a high amount of knowledge about the workings of such techniques. This problem has already been identified in [124, 174], where the creation of benchmarks to compare the performance of different techniques has been labeled as one of the biggest challenges concerning process mining.

To tackle this issue, scholars have proposed frameworks [221] that help to compare the performance of process discovery techniques, whereas others have focused on benchmarking the quality of those algorithms [47]. However, one important aspect to remark is that previous studies focused more on the algorithms themselves together with their general performance, without going into much detail on how they are influenced based on the characteristics of a given event log (the starting point of analysis). In [124], the importance of these characteristics is also mentioned. As such, this section aims to present a benchmarking study of process discovery techniques which will help to understand the relation between event log characteristics and the performance of a process discovery algorithm and can aid process mining practitioners to make more well-educated choices about the most suitable technique for different scenarios. To do so, an in-depth analysis on the influence that event log characteristics have on the performance of process discovery techniques is presented. In addition, a comparative study between techniques' performances and accuracy in different situations is also carried out.

8.7.1 Related Work

The benchmarking of process discovery techniques is a problem which has not been addressed extensively in the literature [124]. However, the need for such studies is real and motivated by the proliferation of techniques and commercial products that has been occurring recently [174].

The first attempt to come up with a framework that allows to benchmark process discovery techniques was executed by Rozinat et al. [124, 234]. These studies aim to reach a "common evaluation framework" which, under the assumption that the log reflects the original behavior, enables the comparison of a discovered model with the event log from which it has been induced by means of conformance analysis. After the groundwork had been set by these publications, other authors utilized similar procedures to construct their frameworks. Weber et al. [222] design a procedure that evaluates the performance of process mining algorithms against a "known grounded truth", referencing to the original process models, with known constructs, that are used to generate the event logs used for the conformance analysis. Although having introduced concrete experiments on artificial process models, which allowed the comparison between the performances of different mining algorithms, only a small number of techniques and models were evaluated (also taking into account a relatively small number of structural properties).

After arguing that the procedure proposed to that date was time consuming and computationally demanding (as this effort is proportional to the number of algorithms and metrics considered), Wang et al. [220, 221] suggest a different approach. Their proposed methodology allows for a more scalable method as the evaluation of the algorithms' performance is done on a fraction of the process models of a given company, making it possible to choose the most suitable technique without having to mine all processes available. Finally, De Weerdt et al. [47] include real-life event logs in their study, thus being the first authors to consider such kind of logs. In addition, they propose the "F-score" metric as a valid approach when evaluating the combination of different accuracy dimensions.

8.7.2 Methodology

As stated before, the motivation for our study is to understand the effect of the characteristics of event logs on the performance of process discovery techniques, allowing us to pinpoint practical guidelines that helps one making problem-tailored decisions when opting for a process discovery technique.

An experiment was designed comprising of the following phases:

1. Phase 1: creation of synthetic process models and generation of event logs.
2. Phase 2: mining of generated event logs using broad collection of process discovery techniques.

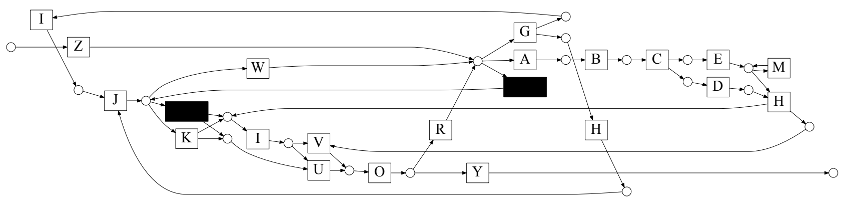


Figure 8.5: Eight process models with increasing complexity levels were designed for use in the benchmarking study. The process model shown here is the most complex and contains all included characteristics.

3. Phase 3: evaluation of the quality of the mined models according to the four process model quality dimensions [124], using a collection of published conformance checking metrics.

Before presenting the results of the study, the three phases in the experimental setup are discussed in more detail in the following subsections.

8.7.2.1 Phase 1: Synthetic Model Construction and Event Log Generation

Eight process models with increasing complexity levels were designed. The following structural patterns, ordered by complexity, were taken into account: sequence, choice (i.e.: XOR split/join), parallelism (i.e.: concurrency, AND split/join), loops (repetitions, recurrence), duplicate tasks (activities in the model bearing the same label), non-free choice constructs (i.e. history dependent choices) and finally, nested loops. For each characteristic listed above, a process model was constructed containing this characteristic, together with all characteristics from less complex models, meaning that model eight (shown in Figure 8.5), the most complex model designed, contains all listed characteristics.

Artificial event logs were generated (using the event log generator discussed above) based on the eight designed process models with different non-structural characteristics, namely presence of noise (removed events, swapped events and inserted events) and the size of the event log. Table 8.1 shows an overview of the properties of the generated event logs for each process model.

Table 8.1: For each designed process model, four event logs were generated of different size, with and without noise to represent real-life scenario's.

	Small	Medium	Medium with Noise	Large with Noise
Number of events	500	5000	5000	50000
Noise	No	No	Yes	Yes

8.7.2.2 Phase 2: Process Model Discovery

Twelve process discovery techniques are included in the study: Alpha Miner [55], Alpha Miner+ [58], Alpha Miner++ [68], Heuristics Miner [86], Genetic Miner [65], DT (Duplicate Tasks) Genetic Miner [66], DWS Miner [62], AGNEs Miner [2], the Evolutionary Process Tree Miner [113], Causal Miner [43], ILP Miner [77] and TS (Transition System) Miner [79]. The list of mining algorithms was constructed in such way so that all included process discovery techniques satisfy the following requirements. First, their result should be represented in the form of a Petri net or be transformable into one for the sake of clear and unambiguous comparability. Second, algorithms should be publicly available and implemented in the tools used for research, i.e. ProM 5 and ProM 6 [46], which were applied to mine the models from the constructed event logs.

Concerning configuration options, parameters for each discovery technique were mostly kept to the default options. However, modifications were made for the Heuristics Miner and Causal Miner. More precisely, the option "mine long distance dependencies" was enabled for both, in order to enable to discovery of these constructs. Also, both for Genetic Miner and DT Genetic Miner, the "population size" was set to 10 and 100 for high and small/medium complexity logs respectively, as to keep running time under control. For the Process Tree Miner—another evolutionary algorithm—"population size" was also set to 10 and the "maximum number of generations" to 50.

8.7.2.3 Phase 3: Conformance Checking and Statistical Analysis

In the third and final phase of the experimental setup, an exhaustive conformance analysis is performed using a large amount of conformance checking metrics in order to assess the quality of different mined process models in respect with the generated event logs based on four quality dimensions. Next, the results are analyzed and compared using an exhaustive set of statistical tests.

Conformance Checking As we have discussed throughout this dissertation, conformance checking techniques demonstrate the representativeness of the discovered process model and the behavior presented in the event log. Conformance is typically measured across the following four quality dimensions [124]:

- Fitness: the discovered model should allow for the behavior seen in the event log.
- Precision: the discovered model should not allow for behavior completely unrelated to what was seen in the event log.
- Generalization: the discovered model should generalize beyond the example behavior seen in the event log.
- Simplicity: the discovered model should be as simple as possible.

An overview of conformance checking metrics included in this study for each quality dimension is provided in Table 8.2. The CoBeFra conformance checking benchmarking suite was applied in order to facilitate the execution of the conformance checking phase in our study. It is important to note that virtually all metric included in the study return a result in the range of [0, 1]. Most simplicity metrics, however, except for Advanced Structural Appropriateness, return an absolute value, indicating for instance a count. Therefore, we normalize these values before performing the statistical analysis so that 0 corresponds with the minimum (lowest) score obtained and 1 with the maximum (highest score).

Statistical Analysis After conformance analysis is performed between all mined models and event logs, a set of statistical techniques is applied to evaluate the results, enabling a robust and mathematical rigorous way of comparing the performance of different process discovery techniques (both in general and taking into account specific event log characteristics).

In a first step, a one-way repeated measures ANOVA (Analysis of Variance) test is executed, which is used to analyze the differences between the metrics results for each quality dimension, that is, to assess whether the different metrics “agree” on their result.

Next, a regression analysis is performed to discover the relation between structural process model and event log properties and the performance of process

Table 8.2: Overview of conformance checking metrics included in the experimental setup.

Name	Authors	Quality Dimension
Fitness	Rozinat et al. [99]	Fitness
Proper Completion	Rozinat et al. [99]	
Alignment Based Fitness	Adriansyah et al. [110]	
Behavioral Profile Conformance	Weidlich et al. [104]	
Behavioral Recall	Goedertier et al. [2, 235]	
Advanced Behavioral Appropriateness	Rozinat et al. [99]	Precision
ETC Precision	Muñoz-Gama et al. [105]	
Alignment Based Precision	Adriansyah et al. [109]	
One Align Precision	Adriansyah et al. [112]	
Best Align Precision	Adriansyah et al. [112]	
Behavioral Precision	De Weerdt et al. [107]	
Alignment Based Probabilistic Generalization	Adriansyah et al. [109]	Generalization
Advanced Structural Appropriateness	Rozinat et al. [99]	Simplicity
Average Node Arc Degree	Mendling et al. [114, 236]	
Count of Arcs	Mendling et al. [114, 236]	
Count of Cut Vertices	Mendling et al. [114, 236]	
Count of Nodes	Mendling et al. [114, 236]	
Count of Places	Mendling et al. [114, 236]	
Count of Transitions	Mendling et al. [114, 236]	
Weighted P/T Average Arc Degree	Mendling et al. [114, 236]	

discovery techniques, this for each discovery technique in each quality dimension, so that we can uncover main "driving factors" behind each process discovery technique. Depending on the results of the previous ANOVA test, different response variables are used in the regression test: if the null hypothesis was accepted (no difference in means of metric results), the average of all metrics for one dimension was used as the independent variable (with the different characteristics of logs and models as dependent variables). If the null hypothesis was rejected, the independent variable is based on the result of a carefully selected metric (or metrics) for the four quality dimensions, as it would be unfeasible to incorporate all metrics together with each discovery technique.

Finally, a third test aims to compare the performance of different process discovery techniques in a robust manner. We apply a non-parametric statistical approach towards the comparison of process discovery techniques as outlined in [47, 237], encompassing the execution of a non-parametric Friedman test followed by an appropriate post-hoc test. These non-parametric tests assume ranked performances rather than actual performance estimates, calculating the average ranks per treatment, under the null hypothesis of no significant differences between treatments, i.e. process discovery algorithms. If the null hypothesis of equal performance across all treatments is rejected by the Friedman test, we proceed with a post hoc Bonferroni-Dunn procedure, comparing one treatment (the control treatment) to all others. This test thus compares the different process discovery techniques to the best performing one and assesses whether or not the performances are similar.

8.7.3 Results

In this section the results derived from the different analysis that were performed are presented and dissected in order to come up with two major contributions: first, an understanding of the correlation between event log characteristics and technique's performance and second: a comparative performance assessment of available process discovery techniques.

8.7.3.1 Statistical Results

First, the results of the included statistical tests are provided. This allows both to address the major goals of this research and to check the significance analysis of the results. A ranking of process discovery techniques according to their conformance performance will be presented, followed by an analysis of run time performance. Finally, the outcome of this benchmarking study is exposed as a set of practical guidelines that are listed at the end of this section.

8.7.3.2 Comparing Conformance Checking Metric Result Similarity: ANOVA

The first statistical test performed was a one-way repeated measures ANOVA (analysis of variance) test, with the goal of evaluating if the different metrics within the same dimension provide similar results. The null hypothesis is that averages of all metrics (within one dimension) are equal, i.e. meaning that the metrics within this quality dimension "agree" with one another. This hypothesis was rejected for all dimensions; all p-values were below a significance level of 0.05. A Mauchly test for sphericity [238] was performed followed by Greenhouse-Geisser [239] and Huynh-Feldt [240] corrections to obtain valid F-ratios.

8.7.3.3 Driving Factors behind Process Discovery Algorithms: Regression

A linear regression analysis was performed to find the correlation between conformance checking metrics (dependent variable) and characteristics of both event logs and models (independent variables) for each of the four quality dimensions and for every process discovery technique. As the previous test showed that there was a significant variation between results from different metrics, we decide on conformance checking metrics for each of the four quality dimensions to configure the dependent variable. This decision was driven by the following criteria: first, the conformance metric should not result in too many missing values, indicating errors during execution or exceeding a run time limit of 24 hours. Second, the metric should not always result in very high or low values. Third, metrics which are described in more recent literature are preferred above earlier approaches. Based on these criteria, the following metrics were chosen to represent each quality dimension:

- Fitness: *Alignment Based Fitness* (symbol used: F^{a}, Adriansyah et al. [110]) and *Behavioral Recall* (symbol used: F^{b}, Goedertier et al. [2, 235])
- Precision: *One Align Precision* (symbol used: P^{a}, Adriansyah et al. [112]) and *Weighted Behavioral Precision* (symbol used: P^{b})
- Generalization: *Alignment Based Probabilistic Generalization* (symbol used: G^{a}, Adriansyah et al. [109])
- Simplicity: *Weighted P/T Average Arc Degree* (symbol used: S, Mendling et al., Sánchez-González et al. [114, 236])

The results show that the values obtained by the conformance checking metrics are indeed correlated with several event log characteristics. Table 8.3 provides a summarized overview of the driving characteristics for all process discovery algorithms for the four quality dimensions.

8.7.3.4 Comparative Performance Analysis of Process Discovery Algorithms: Friedman and Bonferroni-Dunn

A Friedman test is applied in order to determine whether there is a significant difference in the performance of the discovery techniques, based on the ranking of their quality results, using the same conformance checking metric for each quality dimension as for the regression analysis. The results show that techniques do not perform equivalently (null hypothesis rejected), this for all four quality dimensions (using a 95% confidence level and using the conformance

Table 8.3: Results of regression analysis, indicating the manner by which the different process discovery techniques are influenced by event log and process model characteristics.

Characteristic	Alpha Miner	Alpha Miner+	Alpha Miner++	Heuristics Miner	Genetic Miner	DT Genetic Miner
Choice	$S\downarrow$	$P^b\downarrow$ $S\downarrow$	$F^b\uparrow$ $P^b\uparrow$	$P^b\uparrow$	$F^a\downarrow$ $P^a\downarrow$	$F^a\downarrow$ $P^b\downarrow$ $G\uparrow$
Parallelism		$P^b\downarrow$	$P^b\uparrow$	$P^b\uparrow$		
Loop	$F^a\downarrow F^b\downarrow$ $P^b\downarrow$ $S\uparrow$	$F^b\downarrow$ $S\uparrow$	$F^a\downarrow F^b\downarrow$ $P^a\downarrow P^b\downarrow$ $G\downarrow$ $S\uparrow$	$S\uparrow$	$F^a\uparrow$ $P^a\uparrow$	$F^b\uparrow$
Invisible Tasks						
Duplicate Tasks				$F^a\downarrow$ $P^a\downarrow$		$F^a\uparrow F^b\uparrow$ $P^a\uparrow P^b\uparrow$ $G\downarrow$
Non-free Choice	$P^a\downarrow P^b\downarrow$ $G\downarrow$	$P^b\uparrow$ $F^b\uparrow$		$F^a\downarrow F^b\uparrow$	$F^a\uparrow$ $P^a\uparrow$	$S\uparrow$
Nested Loop	$F^a\downarrow F^b\downarrow$ $P^a\uparrow$ $S\uparrow$	$F^b\downarrow$ $S\uparrow$	$F^b\downarrow$ $P^b\downarrow$ $S\uparrow$	$F^a\downarrow$		
Number of Traces				$P^b\downarrow$	$F^a\uparrow$ $P^a\uparrow$	$F^a\uparrow$ $G\downarrow$ $S\downarrow$
Number of Distinct Traces			$P^a\uparrow$		$F^a\uparrow$	
Number of Events				$P^b\uparrow$	$F^a\downarrow$ $P^a\downarrow$	$F^a\downarrow F^b\uparrow$ $G\uparrow$ $S\uparrow$
Minimum Trace Length	$F^a\downarrow$ $P^a\uparrow$ $S\uparrow$	$F^b\downarrow$ $P^b\downarrow$ $S\uparrow$	$F^b\downarrow$ $P^b\downarrow$ $S\uparrow$		$F^a\uparrow$ $P^a\uparrow$	$F^a\uparrow F^b\downarrow$ $G\uparrow$
Average Trace Length	$F^a\uparrow F^b\uparrow$ $P^a\downarrow$ $S\downarrow$	$F^b\uparrow$ $P^b\downarrow$ $S\downarrow$	$F^b\uparrow$ $P^b\uparrow$ $S\downarrow$	$F^a\uparrow$	$F^a\downarrow$ $P^a\downarrow$	
Maximum Trace Length	$F^b\uparrow$ $S\downarrow$	$F^b\uparrow$ $S\downarrow$	$F^b\uparrow$ $S\downarrow$	$F^b\downarrow$		$F^b\downarrow$
Noise	$S\uparrow$	$F^b\downarrow$ $P^b\uparrow$	$S\uparrow$	$F^b\downarrow$ $P^b\downarrow$ $S\uparrow$	$F^a\uparrow$	
Number of Activities	$F^a\downarrow F^b\downarrow$ $P^a\uparrow$ $S\uparrow$	$F^b\downarrow$ $P^b\downarrow$ $S\uparrow$	$F^b\downarrow$ $P^b\downarrow$ $S\uparrow$	$F^a\downarrow$ $P^b\downarrow$ $S\uparrow$	$P^a\uparrow$	$F^a\downarrow F^b\downarrow$ $P^a\downarrow P^b\downarrow$ $G\downarrow$

Table 8.3 (continued): Results of regression analysis.

DWS Miner	AGNEs Miner	TS Miner	ILP Miner	Causal Miner	Process Tree Miner	Characteristic
$F^a \downarrow$ $F^b \uparrow$		$P^a \uparrow$ $S \downarrow$	$F^a \uparrow$ $P^a \uparrow$ $G \uparrow$			Choice
$F^b \uparrow$						Parallelism
$F^a \uparrow$ $P^b \downarrow$ $S \uparrow$	$S \uparrow$	$F^a \downarrow F^b \downarrow$ $P^a \uparrow P^b \downarrow$ $S \downarrow$	$F^a \downarrow$ $P^a \downarrow P^b \downarrow$ $G \downarrow$	$S \uparrow$		Loop
						Invisible Tasks
$F^a \downarrow$ $P^a \downarrow$	$P^b \downarrow$	$F^a \uparrow F^b \uparrow$ $P^b \downarrow$ $S \downarrow$	$F^a \downarrow$ $G \downarrow$			Duplicate Tasks
$F^a \uparrow F^b \uparrow$ $P^b \downarrow$	$F^b \downarrow$ $P^b \downarrow$	$F^b \downarrow$ $P^a \downarrow P^b \downarrow$ $S \uparrow$	$F^a \downarrow$ $P^a \downarrow$ $G \downarrow$	$F^a \downarrow F^b \downarrow$ $P^b \downarrow$ $S \uparrow$		Non-free Choice
$F^a \uparrow F^b \downarrow$ $P^b \downarrow$ $S \uparrow$	$F^b \downarrow$ $S \uparrow$	$P^b \downarrow$ $S \downarrow$	$F^a \downarrow$ $P^a \downarrow$ $G \downarrow$	$S \uparrow$		Nested Loop
	$F^b \uparrow$				$S \uparrow$	Number of Traces
$F^a \uparrow F^b \downarrow$ $P^b \downarrow$	$P^b \downarrow$ $S \uparrow$	$P^a \uparrow$ $F^a \uparrow$	$G \uparrow$ $S \uparrow$	$F^a \uparrow$		Number of Distinct Traces
	$F^b \downarrow$ $P^a \uparrow$				$S \uparrow$	Number of Events
$F^a \uparrow F^b \downarrow$ $P^a \downarrow P^b \downarrow$ $S \uparrow$	$F^b \downarrow$	$F^a \downarrow F^b \downarrow$ $P^a \downarrow P^b \downarrow$ $S \uparrow$	$F^a \downarrow$ $P^a \downarrow$ $G \downarrow$ $S \uparrow$	$F^a \uparrow$		Minimum Trace Length
$F^a \downarrow F^b \uparrow$ $P^b \uparrow$ $S \downarrow$	$F^b \uparrow$		$F^a \uparrow$ $P^a \uparrow$ $G \uparrow$ $S \downarrow$	$F^a \downarrow$		Average Trace Length
	$S \downarrow$		$S \uparrow$	$S \downarrow$		Maximum Trace Length
$P^a \downarrow$ $S \uparrow$			$P^b \downarrow$ $S \uparrow$			Noise
$F^a \uparrow F^b \downarrow$ $P^b \downarrow$ $S \uparrow$	$F^b \downarrow$ $S \uparrow$	$P^a \downarrow$ $S \uparrow$	$F^a \downarrow$ $P^a \downarrow$ $G \downarrow$ $S \uparrow$	$F^a \uparrow$ $S \uparrow$		Number of Activities

checking values obtained by the metrics selected in Subsection 8.7.3.3 to establish the rankings). We thus perform post hoc Bonferroni-Dunn tests for all quality dimensions.

Table 8.4 depicts an overview of the obtained results. For each quality dimension, the average rank for each process discovery technique is depicted (a higher rank indicates better performance), with the best performing technique shown in bold and underlined. Techniques which do not differ significantly at the 95% confidence level from the best performing technique are shown in bold. From this table, a tradeoff between fitness/generalization on the one hand and simplicity/precision on the other hand becomes apparent for many discovery techniques. These results point to the difficulty of having process discovery techniques which perform well for all four quality dimensions [218]. In addition, observe the differences between the rankings obtained by different conformance checking metrics within the same quality dimension. This again points to the fact that metrics do not necessary agree on the way they perform their quality assessment. As such, many opportunities for improvement and further research remain in this area.

8.7.3.5 Evaluation of Run Time

Although not explicitly included in this study, for many real-life applications, the time required to perform a discovery procedure is a critical issue. Motivated by this fact, the run time for each algorithm was recorded. Although the running time was not included in a statistical test as-is, the average speed of each discovery technique was taken into account to formulate the practical guidelines below.

8.7.4 Recommendations towards Choosing a Process Discovery Technique

Taking into consideration both general performance results (from the Friedman and Bonferroni-Dunn tests), the influence of event log and process model based characteristics (from the regression analysis) and timing issues, initial, general recommendations for choosing an appropriate process discovery technique are formulated.

Table 8.4: Bonferonni-Dunn rankings for process discovery techniques. For each quality dimension, the average rank for each process discovery technique is depicted (1 being the lowest rank). Techniques which do not differ significantly at the 95% confidence level from the best performing technique are in italics type.

Fitness		Generalization
Alignment Based Fitness	Behavioral Recall	Alignment Based Pr.Generalization
ILP Miner (9.75)	*ILP Miner (11.11)*	*AGNEs Miner (9.36)*
AGNEs Miner (9.64)	*Heuristics Miner (9.61)*	*ILP Miner (9.34)*
Causal Miner (9.31)	*Alpha Miner+ (9.45)*	*TS Miner (8.53)*
DWS Miner (8.72)	*AGNEs Miner (8.89)*	*Heuristics Miner (8.42)*
Heuristics Miner (8.59)	*DWS Miner (8.77)*	*DWS Miner (8.28)*
TS Miner (8.48)	TS Miner (6.69)	*Causal Miner (7.95)*
Region Miner (6.80)	Genetics Miner (6.56)	*Alpha Miner (7.22)*
Process Tree Miner (6.34)	Alpha Miner (6.28)	*Process Tree Miner (7.06)*
Alpha Miner (6.30)	Alpha Miner++ (6.17)	Alpha Miner++ (6.28)
Genetics Miner (5.77)	Causal Miner (5.28)	Genetics Miner (6.00)
Alpha Miner++ (5.11)	DT Genetics Miner (4.77)	Region Miner (5.83)
DT Genetics Miner (4.83)	Region Miner (4.08)	DT Genetics Miner (4.80)
Alpha Miner+ (1.36)	Process Tree Miner (3.34)	Alpha Miner+ (1.92)

Simplicity	Precision	
Weighted P/T Average Arc Degree	One Align Precision	Behavioral Precision
Alpha Miner+ (11.07)	*AGNEs Miner (10.34)*	*DWS Miner (10.09)*
DT Genetics Miner (8.52)	*DWS Miner (9.20)*	*AGNEs Miner (9.38)*
AGNEs Miner (7.12)	*Alpha Miner (8.75)*	*Causal Miner (8.97)*
Causal Miner (6.93)	*ILP Miner (8.67)*	*Alpha Miner (8.78)*
DWS Miner (6.72)	*Heuristics Miner (8.61)*	*Heuristics Miner (8.44)*
ILP Miner (6.72)	*Causal Miner (7.83)*	*ILP Miner (8.27)*
Alpha Miner++ (5.43)	*TS Miner (7.78)*	Alpha Miner++ (7.45)
Region Miner (5.26)	Alpha Miner++ (7.48)	TS Miner (7.11)
Heuristics Miner (5.24)	Region Miner (6.20)	Genetics Miner (6.30)
Alpha Miner (4.79)	Genetics Miner (5.52)	Process Tree Miner (5.14)
Genetics Miner (4.41)	Process Tree Miner (5.09)	Region Miner (4.11)
Process Tree Miner (4.09)	DT Genetics Miner (4.33)	DT Genetics Miner (4.09)
TS Miner (3.59)	Alpha Miner+ (1.19)	Alpha Miner+ (2.88)

After considering all quality dimensions from an overall perspective, the following process discovery techniques are recommended. First, Heuristics Miner, which offers fast run time with acceptable quality results. Next, DWS Miner, which was found to be somewhat slower but also offers good quality results. Third, ILP Miner offers high quality levels but comes with very high run times and memory requirements as event logs become more complex. Finally, AGNEsMiner also presents a slower run time for large models, but is also able to reach good quality levels.

Naturally, it is possible to fine tune the selection of a discovery algorithm in case the quality dimension of interest (or quality priorities) are known beforehand, this even more so when the conformance checking metric which will be applied is known to the end user as well. Since we do not assume such prior knowledge, we limit our recommendations to a general listing only. Similarly, if the characteristics of the event logs are known beforehand, it also useful to identify which techniques are best able to handle these properties, based on the results shown in Table 8.4. As such, the results presented in this section are preliminary work. As we indicated earlier, there has been a rising interest recently in constructing recommender systems to recommend a process discovery algorithm based on the characteristics derived from a given event log, possibly in combination with user preferences regarding which aspects they assign more importance to (speed, fitness, etc.). We refer to [220, 221, 241, 242] for further reading on the topic of process discovery technique recommendation systems.

Chapter 9

Conclusions

"So now as I'm leaving
I'm weary as hell
The confusion I'm feeling
Ain't no tongue can tell
The words fill my head
And they fall to the floor
If God is on our side
He'll stop the next war."
– Bob Dylan

9.1 Overview

This thesis has outlined a number of novel techniques, implementations and applications in the domain of process mining. In the first part of the dissertation, we have discussed an improved artificial negative event generator, and subsequently enhanced this technique with a scoring mechanism to assign a measure of confidence to induced negative events. These induced negative events were then applied in a conformance checking setting, as they allow us to develop a comprehensive conformance checking framework in line with standard machine learning practices, allowing to assess the recall, precision and generalization of a process model. Finally, we have also indicated how artificial negative events can be applied to uncover implementation problems by using them as highlighters of unobserved behavior.

In the second part of the dissertation, we have outlined a number of additional contributions which are not directly related to the concept of artificial negative events. In particular, a novel heuristic process discovery technique was presented, based on a long lineage of related process discovery algorithms, but with a particular focus on robustness and flexibility. Based on replay strategies developed throughout the first part of the thesis, an event-granular conformance analysis technique was presented which can be applied towards enabling real-time monitoring of business activities. Next, a technique for explaining event log cluster solutions on an instance-granular level was proposed, and finally, a benchmarking framework was developed to enable the automated set-up of large-scale conformance analysis experiments.

We summarize the contributions of our work as follows:

- An *improved artificial negative event induction technique* was introduced, extending earlier approaches [1–3].
- The developed technique was extended based of the concept of a *weighted artificial negative event.* The initial improved generation technique was greatly extended and optimized, and an empirical evaluation experiment has underlined the validity of the scoring metric used towards assigning a weighting to artificial negative events in a manner which is robust to incompleteness and noise.
- Based on the concept of weighted artificial negative events, a comprehensive *conformance checking framework* was presented, which is able to assess the recall, precision and generalization quality dimensions of process models in a robust, integrated, and scalable manner. We have developed a number of replay strategies and negative event evaluation methods to implement the conformance checking framework. In a benchmarking study, our developed metrics were compared with other related works.
- Finally, we have indicated how the concept of artificial negative events can be applied as a *detection mechanism for unobserved behavior.* We show how this can help to highlight implementation problems, both for procedural and declarative process models. This idea forms the basis of a more comprehensive event existence classification framework.

- We have presented *Fodina*, a novel heuristic process discovery technique, based on a long lineage of related process discovery algorithms, but with a particular focus on robustness and flexibility. The technique is able to discover the construct of duplicate activities, and a benchmarking study confirmed the robustness and scalability of the technique for both synthetic and real-life event logs.
- Based on the replay strategies developed throughout the first part of the thesis, an *event-granular conformance analysis technique* was presented. The technique can be applied towards enabling real-time monitoring of business activities. By applying the concept of process model decomposition, the technique is able to localize deviations in a more precise and fine-grained manner and the technique can be ran in a distributed manner.
- We have introduced *SECPI* (Search for Explanations for Clustered Process Instances), a algorithm for explaining event log cluster solutions on an instance-granular level.
- Finally, we have developed and presented *CoBeFra* (Comprehensive Benchmarking Framework): a technique devoted to enable the automated set-up of mass-scale conformance analysis experiments. We have also presented a developed Petri net based event log generation technique which enables to rapidly construct a set of synthetic event logs for use within experimental setups and we outlined the results of a benchmarking study aiming to uncover the relationship between process discovery techniques and event log characteristics.
- All the developed tools and techniques were *implemented* and tested on both synthetic and real-life logs and made available for public use.

9.2 Future Work

Regarding possibilities for future work, we refer to the different chapters in this dissertation to obtain a detailed overview. In general, we can summarize possibilities for future work as follows:

- For the *artificial weighted negative event generation techniques*: experimentation with configuration parameters regarding artificial negative event induction, e.g. the inclusion of a noise-related threshold.

- For the *conformance checking framework based on artificial weighted negative events*: extending the framework so it can be applied on any process model representation language for which event-granular execution semantics can be defined (either heuristic or not).
- For *Fodina*: extending the algorithm so that it takes into account negative events during the discovery phase, e.g. by modifying the dependency metrics. Experiment with additional item-set based techniques for the determination of duplicate activities.
- For the *real-time decomposed conformance analysis*: setting up a case study, adapting the technique so it can deal with an event stream without events carrying a case identifier, adapting the technique so it can be used as a recommendation engine (i.e. enabling "self-healing processes").
- For *SECPI*: experimenting with different attribute templates, and setting up a thorough study to analyze clustering algorithm behavior.
- For *CoBeFra*: offering CoBeFra in a service-oriented manner, investigating how to framework can be adapted towards process discovery, investigating which changes can be made to the ProM framework to allow for better setup of batch experiments.
- For the *Petri net based event log generator*: allowing for more configurable regarding timings and arrival time and duration distributions. The inclusion of event log generation functionality in an existing tool can be considered.

Lastly, a number of published works which came to fruition in the context of this dissertation were not included. The interested reader is referred to the publication list at the end of this dissertation for a detailed overview.

End Matter

List of Figures

List of Tables

Bibliography

[1] Stijn Goedertier, David Martens, Bart Baesens, Raf Haesen, and Jan Vanthienen. Process mining as first-order classification learning on logs with negative events. In ter Hofstede et al. [243], pages 42–53.

[2] Stijn Goedertier, David Martens, Jan Vanthienen, and Bart Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10:1305–1340, 2009.

[3] S. Goedertier. *Declarative Techniques for Modeling and Mining Business Processes.* Phd thesis, Katholieke Universiteit Leuven, Faculty of Business and Economics, Leuven, September 2008.

[4] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes.* Springer, 2011.

[5] T. Hastie, R. Tibshirani, and J. Friedman. *The elements of statistical learning: data mining, inference and prediction (2nd edition).* Springer Science, New York, 2009.

[6] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining.* Addison-Wesley, 2005.

[7] Jan Vom Brocke and Michael Rosemann. *Handbook on Business Process Management: Strategic Alignment, Governance, People and Culture.* Springer, 2010.

[8] wedowebsphere.de. Process mining koryphäe wil van der aalst im gespräch mit wedowebsphere. `http://wedowebsphere.de/news/process-mining-koryph%C3%A4e-wil-van-der-aalst-im-gespr%C3%A4ch-mit-wedowebsphere`, 2014.

[9] Betsi Harris Ehrlich. *Transactional Six Sigma and Lean Servicing: Leveraging Manufacturing Concepts to Achieve World-Class Service.* CRC, 2002.

[10] Motorola Inc. Motorola University, Six Sigma in Action. `http://www.motorola.com/motorolauniversity`, consulted on January 18, 2007, 1986.

[11] P. Nonthaleerak and L. Hendry. Exploring the Six Sigma phenomenon using multiple case study evidence. *International Journal of Operations & Production Management*, 28(3):279–303, 2008.

[12] Geoff Tennant. *Six Sigma: SPC and TQM in manufacturing and services.* Gower Publishing, Ltd., 2001.

[13] Matteo Golfarelli, Stefano Rizzi, and Iuris Cella. Beyond data warehousing: what's next in business intelligence? In Il-Yeol Song and Karen C. Davis, editors, *DOLAP*, pages 1–6. ACM, 2004.

[14] Christophe Mues, Bart Baesens, and Jan Vanthienen. From knowledge discovery to implementation: Developing business intelligence systems using decision tables. In Klaus-Dieter Althoff, Andreas Dengel, Ralph Bergmann, Markus Nick, and Thomas Roth-Berghofer, editors, *Wissensmanagement*, pages 439–443. DFKI, Kaiserslautern, 2005.

[15] D. Martens, J. Vanthienen, S. Goedertier, and B. Baesens. Placing process intelligence within the business intelligence framework. In *Proceedings of the 6th International Conference on Information and Management Sciences (IMS 2007)*, 2007.

[16] Daniela Grigori, Fabio Casati, Malú Castellanos, Umeshwar Dayal, Mehmet Sayal, and Ming-Chien Shan. Business process intelligence. *Computers in Industry*, 53(3):321–343, 2004.

[17] Michael zur Mühlen and Robert Shapiro. Business process analytics. In *Handbook on Business Process Management: Strategic Alignment, Governance, People and Culture.* Springer, 2009.

[18] Edward Peters. Business process analytics – presentation. `www.oc.com`, 2007.

[19] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations.* Morgan Kaufmann, 1999.

[20] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. Knowledge discovery and data mining: Towards a unifying framework. In Evangelos Simoudis, Jiawei Han, and Usama M. Fayyad, editors, *KDD*, pages 82–88. AAAI Press, 1996.

[21] Takashi Washio and Jun Luo, editors. *Emerging Trends in Knowledge Discovery and Data Mining - PAKDD*

2012 International Workshops: DMHM, GeoDoc, 3Clust, and DSDM, Kuala Lumpur, Malaysia, May 29 - June 1, 2012, Revised Selected Papers, volume 7769 of *Lecture Notes in Computer Science*. Springer, 2013.

[22] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Magazine*, 17(3):37–54, 1996.

[23] Hans-Peter Kriegel, Karsten M. Borgwardt, Peer Kröger, Alexey Pryakhin, Matthias Schubert, and Arthur Zimek. Future trends in data mining. *Data Min. Knowl. Discov.*, 15(1):87–97, 2007.

[24] Saso Dzeroski. Multi-relational data mining: an introduction. *SIGKDD Explorations*, 5(1):1–16, 2003.

[25] Daniel A. Keim. Information visualization and visual data mining. *IEEE Trans. Vis. Comput. Graph.*, 8(1):1–8, 2002.

[26] Simeon J. Simoff, Michael H. Böhlen, and Arturas Mazeika, editors. *Visual Data Mining - Theory, Techniques and Tools for Visual Analytics*, volume 4404 of *Lecture Notes in Computer Science*. Springer, 2008.

[27] Daniel A. Keim, Florian Mansmann, Jörn Schneidewind, Jim Thomas, and Hartmut Ziegler. Visual analytics: Scope and challenges. In Simoff et al. [26], pages 76–90.

[28] S. Džeroski and N. Lavrač, editors. *Relational Data Mining*. Springer-Verlag, Berlin, 2001.

[29] Anne Rozinat and Wil M. P. van der Aalst. Decision mining in prom. In Dustdar et al. [244], pages 420–425.

[30] Linh Thao Ly, Stefanie Rinderle, Peter Dadam, and Manfred Reichert. Mining staff assignment rules from event-based data. In Bussler and Haller [245], pages 177–190.

[31] Ricardo Bezerra de Andrade e Silva, Jiji Zhang, and James G. Shanahan. Probabilistic workflow mining. In Robert Grossman, Roberto J. Bayardo, and Kristin P. Bennett, editors, *KDD*, pages 275–284. ACM, 2005.

[32] Martin Kuhlmann, Dalia Shohat, and Gerhard Schimpf. Role mining - revealing business roles for security administration using data mining technology. In *SACMAT*, pages 179–186. ACM, 2003.

[33] Joyce Nakatumba, Michael Westergaard, and Wil M. P. van der Aalst. Generating event logs with workload-dependent speeds from simulation models. In Marko Bajec and Johann Eder, editors, *CAiSE Workshops*, volume 112 of *Lecture Notes in Business Information Processing*, pages 383–397. Springer, 2012.

[34] Anne Rozinat, Moe Thandar Wynn, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Colin J. Fidge. Workflow simulation for operational decision support. *Data Knowl. Eng.*, 68(9):834–850, 2009.

[35] Anne Rozinat, R. S. Mans, Minseok Song, and Wil M. P. van der Aalst. Discovering simulation models. *Inf. Syst.*, 34(3):305–327, 2009.

[36] M. H. Jansen-vullers and M. Netjes. Business process simulation - a tool survey. In *In Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN*, 2006.

[37] J. Nakatumba, A. Rozinat, and N. Russell. Business process simulation: How to get it right. In *International Handbook on Business Process Management*. Springer-Verlag, 2008.

[38] Tadao Murata. Petri Nets: Properties, Analysis and Applications. In *Proceedings of the IEEE*, volume 77-4, pages 541–580, April 1989.

[39] James Lyle Peterson. *Petri Net Theory and the Modeling of Systems*. New Jersey: Prentice-Hall, Inc., 1981.

[40] Wil M. P. van der Aalst. Verification of workflow nets. In Pierre Azéma and Gianfranco Balbo, editors, *ICATPN*, volume 1248 of *Lecture Notes in Computer Science*, pages 407–426. Springer, 1997.

[41] Wil M. P. van der Aalst. The application of petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[42] W. M. P. van der Aalst, K. M. van Hee, and G. J. Houben. Modelling and analysing workflow using a Petri-net based approach. In G. De Michelis, C. Ellis, and G. Memmi, editors, *Proceedings of the 2nd Workshop on Computer-Supported Cooperative Work, Petri nets and related formalisms*, pages 31–50, 1994.

[43] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Causal nets: A modeling language tailored towards process discovery. In Joost-Pieter Katoen and Barbara König, editors, *CONCUR*, volume 6901 of *Lecture Notes in Computer Science*, pages 28–42. Springer, 2011.

[44] A.J.M.M. Weijters, W. van der Aalst, and A. Alves de Medeiros. Process mining with the heuristicsminer algorithm. BETA working paper series 166, TU Eindhoven, 2006.

[45] Boudewijn F. van Dongen, Ana Karla A. de Medeiros, H. M. W. Verbeek, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The prom framework: A new era in process mining tool support. In Ciardo and Darondeau [246], pages 444–454.

[46] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Christian W. Günther, Anne Rozinat, Eric Verbeek, and Ton Weijters. Prom: The process mining toolkit. In Ana Karla A. de Medeiros and Barbara Weber, editors, *BPM (Demos)*, volume 489 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2009.

[47] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. A multi-dimensional quality assessment of state-of-the-art process discovery algorithms using real-life event logs. *Inf. Syst.*, 37(7):654–676, 2012.

[48] Jonathan E. Cook and Alexander L. Wolf. Automating process discovery through event-data analysis. In Dewayne E. Perry, Ross Jeffrey, and David Notkin, editors, *ICSE*, pages 73–82. ACM, 1995.

[49] Jonathan E. Cook and Alexander L. Wolf. Discovering models of software processes from event-based data. *ACM Trans. Softw. Eng. Methodol.*, 7(3):215–249, 1998.

[50] Rakesh Agrawal, Dimitrios Gunopulos, and Frank Leymann. Mining process models from workflow logs. In Hans-Jörg Schek, Fèlix Saltor, Isidro Ramos, and Gustavo Alonso, editors, *EDBT*, volume 1377 of *Lecture Notes in Computer Science*, pages 469–483. Springer, 1998.

[51] Anindya Datta. Automating the discovery of as-is business process models: Probabilistic and algorithmic approaches. *Information Systems Research*, 9(3):275–301, 1998.

[52] Heikki Mannila and Christopher Meek. Global partial orders from sequential data. In Raghu Ramakrishnan, Salvatore J. Stolfo, Roberto J. Bayardo, and Ismail Parsa, editors, *KDD*, pages 161–168. ACM, 2000.

[53] Guido Schimm. Process miner - a tool for mining process schemes from event-based data. In Sergio Flesca, Sergio Greco, Nicola Leone, and Giovambattista Ianni, editors, *JELIA*, volume 2424 of *Lecture Notes in Computer Science*, pages 525–528. Springer, 2002.

[54] Guido Schimm. Mining exact models of concurrent workflows. *Computers in Industry*, 53(3):265–281, 2004.

[55] Wil M. P. van der Aalst, Ton Weijters, and Laura Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[56] A.J.M.M Weijters and W.M.P. van der Aalst. Process mining. discovering workflow models from event-based data. In *IEEE Congress on Evolutionary Computation*, pages 283–290, 2001.

[57] A. J. M. M. Weijters and Wil M. P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162, 2003.

[58] A. Alves de Medeiros, B. van Dongen, W. van der Aalst, and A.J.M.M. Weijters. Process mining: Extending the alpha-algorithm to mine short loops. BETA working paper series 113, TU Eindhoven, 2004.

[59] Joachim Herbst and Dimitris Karagiannis. Workflow mining with inwolve. *Computers in Industry*, 53(3):245–264, 2004.

[60] B. F. van Dongen and W. M. P. van der Aalst. Multi-phase process mining: Aggregating instance graphs into EPCs and Petri nets. In *Proceedings of the 2nd International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management (PNCWB)*, 2005.

[61] Walid Gaaloul, Karim Baïna, and Claude Godart. Towards mining structural workflow patterns. In Kim Viborg Andersen, John K. Debenham, and Roland Wagner, editors, *DEXA*, volume 3588 of *Lecture Notes in Computer Science*, pages 24–33. Springer, 2005.

[62] Gianluigi Greco, Antonella Guzzo, Luigi Pontieri, and Domenico Saccà. Discovering expressive process models by clustering log traces. *IEEE Trans. Knowl. Data Eng.*, 18(8):1010–1027, 2006.

[63] Laura Maruster, A. J. M. M. Weijters, Wil M. P. van der Aalst, and Antal van den Bosch. A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Min. Knowl. Discov.*, 13(1):67–87, 2006.

[64] Hugo M. Ferreira and Diogo R. Ferreira. An integrated life cycle for workflow management based on learning and planning. *Int. J. Cooperative Inf. Syst.*, 15(4):485–505, 2006.

[65] Ana Karla A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. Genetic process mining: an experimental evaluation. *Data Min. Knowl. Discov.*, 14(2):245–304, 2007.

[66] A. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, TU Eindhoven, 2006.

[67] Christian W. Günther and Wil M. P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In Alonso et al. [247], pages 328–343.

[68] Lijie Wen, Wil M. P. van der Aalst, Jianmin Wang, and Jiaguang Sun. Mining process models with non-free-choice constructs. *Data Min. Knowl. Discov.*, 15(2):145–180, 2007.

[69] Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. Inducing declarative logic-based models from labeled traces. In Alonso et al. [247], pages 344–359.

[70] Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Mining taxonomies of process models. *Data Knowl. Eng.*, 67(1):74–102, 2008.

[71] A. Rozinat, M. Veloso, and W. M. P. van der Aalst. Using hidden markov models to evaluate the quality of discovered process models. BPM Center Report BPM-08-10, BPMcenter.org, 2008.

[72] Gil Aires da Silva and Diogo R Ferreira. Applying hidden markov models to process mining. In *Sistemas e Tecnologias de Informação: Actas da 4ª Conferência Ibérica de Sistemas e Tecnologias de Informação, AISTI/FEUP/UPF*, 2009.

[73] Jonas Poelmans, Guido Dedene, Gerda Verheyden, Herman Van der Mussele, Stijn Viaene, and Edward M. L. Peters. Combining business process and data discovery techniques for analyzing and improving integrated care pathways. In Petra Perner, editor, *ICDM*, volume 6171 of *Lecture Notes in Computer Science*,

pages 505–517. Springer, 2010.

[74] Lijie Wen, Jianmin Wang, Wil M. P. van der Aalst, Biqing Huang, and Jiaguang Sun. A novel approach for process mining based on event types. *J. Intell. Inf. Syst.*, 32(2):163–190, 2009.

[75] Francesco Folino, Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Discovering expressive process models from noised log data. In Bipin C. Desai, Domenico Saccà, and Sergio Greco, editors, *IDEAS*, ACM International Conference Proceeding Series, pages 162–172. ACM, 2009.

[76] Diogo R. Ferreira and Daniel Gillblad. Discovering process models from unlabelled event logs. In Dayal et al. [248], pages 143–158.

[77] Jan Martijn E. M. van der Werf, Boudewijn F. van Dongen, Cor A. J. Hurkens, and Alexander Serebrenik. Process discovery using integer linear programming. *Fundam. Inform.*, 94(3-4):387–412, 2009.

[78] W. M. P. van der Aalst, V. Rubin, B. F. van Dongen, E. Kindler, , and C. W. Günther. Process mining: A two-step approach using transition systems and regions. BPM-06-30, BPM Center Report, 2006.

[79] Wil M. P. van der Aalst, Vladimir Rubin, H. M. W. Verbeek, Boudewijn F. van Dongen, Ekkart Kindler, and Christian W. Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software and System Modeling*, 9(1):87–111, 2010.

[80] Josep Carmona, Jordi Cortadella, and Michael Kishinevsky. New region-based algorithms for deriving bounded petri nets. *IEEE Trans. Computers*, 59(3):371–384, 2010.

[81] Josep Carmona and Jordi Cortadella. Process mining meets abstract interpretation. In José L. Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *ECML/PKDD (1)*, volume 6321 of *Lecture Notes in Computer Science*, pages 184–199. Springer, 2010.

[82] F.M. Maggi, A.J. Mooij, and W.M.P. van der Aalst. User-Guided Discovery of Declarative Process Models. In *2011 IEEE Symposium on Computational Intelligence and Data Mining*. 2011.

[83] Fabrizio M. Maggi, R.P. Jagadeesh Chandra Bose, and Wil M.P. van der Aalst. Efficient discovery of understandable declarative process models from event logs. In Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *Advanced Information Systems Engineering*, volume 7328 of *Lecture Notes in Computer Science*, pages 270–285. Springer Berlin Heidelberg, 2012.

[84] Andrea Burattin and Alessandro Sperduti. Heuristics miner for time intervals. In *ESANN*, 2010.

[85] Andrea Burattin and Alessandro Sperduti. Automatic determination of parameters' values for heuristics miner++. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[86] A. J. M. M. Weijters and J. T. S. Ribeiro. Flexible heuristics miner (fhm). In *CIDM* [249], pages 310–317.

[87] Andrea Burattin, Alessandro Sperduti, and Wil M. P. van der Aalst. Heuristics miners for streaming event data. *CoRR*, abs/1212.6383, 2012.

[88] Wil M. P. van der Aalst, Joos C. A. M. Buijs, and Boudewijn F. van Dongen. Towards improving the representational bias of process mining. In Karl Aberer, Ernesto Damiani, and Tharam S. Dillon, editors, *SIMPDA*, volume 116 of *Lecture Notes in Business Information Processing*, pages 39–54. Springer, 2011.

[89] Marc Solé and Josep Carmona. An smt-based discovery algorithm for c-nets. In Haddad and Pomello [250], pages 51–71.

[90] Fluxicon. Disco – discover your processes. `http://fluxicon.com/disco`, 2013.

[91] C. Di Ciccio and M. Mecella. A two-step fast algorithm for the automated discovery of declarative workflows. In *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*, pages 135–142, April 2013.

[92] M. Westergaard, C. Stahl, and H.A. Reijers. Unconstrainedminer: efficient discovery of generalized declarative process models. *BPM Center Report BPM-13-28, BPMcenter.org*, page 28, 2013.

[93] Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Bart Baesens. Declarative process discovery with evolutionary computing. In *IEEE Congress on Evolutionary Computation*, pages 1–8, 2014.

[94] Sander J.J. Leemans, Dirk Fahland, and WilM.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*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer Berlin Heidelberg, 2013.

[95] Fabrizio Maria Maggi, Tijs Slaats, and HajoA. Reijers. The automated discovery of hybrid processes. In Shazia Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 392–399. Springer International Publishing, 2014.

[96] Borja Vázquez-Barreiros, Manuel Mucientes, and Manuel Lama. A genetic algorithm for process discovery guided by completeness, precision and simplicity. In Shazia Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 118–133. Springer International Publishing, 2014.

[97] Raffaele Conforti, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Beyond tasks and gateways: Discovering bpmn models with subprocesses, boundary events and activity markers. In Shazia

Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 101–117. Springer International Publishing, 2014.

[98] Johannes De Smedt, Jochen De Weerdt, and Jan Vanthienen. Multi-paradigm process mining: Retrieving better models by combining rules and sequences. In *22nd International Conference on Cooperative Information Systems (CoopIS)*. Springer, 2014.

[99] A. Rozinat, M. Veloso, and W. van der Aalst. Evaluating the quality of discovered process models. In *Proceedings of IPM 2008 induction of process models (ECML PKDD 2008)*, pages 45–52, 2008.

[100] Anne Rozinat and Wil M. P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Inf. Syst.*, 33(1):64–95, 2008.

[101] Seppe vanden Broucke, Jochen De Weerdt, Jan Vanthienen, and Bart Baesens. Determining process model precision and generalization with weighted artificial negative events. *IEEE Transactions on Knowledge and Data Engineering*, 99(PrePrints):1, 2013.

[102] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, Jan Mendling, and Mathias Weske. Process compliance analysis based on behavioural profiles. *Inf. Syst.*, 36(7):1009–1025, 2011.

[103] Matthias Weidlich, Jan Mendling, and Mathias Weske. Efficient consistency measurement based on behavioral profiles of process models. *IEEE Trans. Software Eng.*, 37(3):410–429, 2011.

[104] Matthias Weidlich, Artem Polyvyanyy, Nirmit Desai, and Jan Mendling. Process compliance measurement based on behavioural profiles. In Barbara Pernici, editor, *CAiSE*, volume 6051 of *Lecture Notes in Computer Science*, pages 499–514. Springer, 2010.

[105] Jorge Munoz-Gama and Josep Carmona. A fresh look at precision in process conformance. In Hull et al. [251], pages 211–226.

[106] Jorge Munoz-Gama and Josep Carmona. Enhancing precision in process conformance: Stability, confidence and severity. In *CIDM* [249], pages 184–191.

[107] Jochen De Weerdt, Manu De Backer, Jan Vanthienen, and Bart Baesens. A robust f-measure for evaluating discovered process models. In *CIDM* [249], pages 148–155.

[108] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Conformance checking using cost-based fitness analysis. In *EDOC*, pages 55–64. IEEE Computer Society, 2011.

[109] Wil M. P. van der Aalst, Arya Adriansyah, and Boudewijn F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisc. Rew.: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[110] Arya Adriansyah, Natalia Sidorova, and Boudewijn F. van Dongen. Cost-based fitness in conformance checking. In Benoît Caillaud, Josep Carmona, and Kunihiko Hiraishi, editors, *ACSD*, pages 57–66. IEEE, 2011.

[111] Arya Adriansyah, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Towards robust conformance checking. In zur Muehlen and Su [252], pages 122–133.

[112] Arya Adriansyah, Jorge Munoz-Gama, Josep Carmona, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. Alignment based precision checking. In Rosa and Soffer [253], pages 137–149.

[113] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. A genetic algorithm for discovering process trees. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[114] Jan Mendling, Gustaf Neumann, and Wil M. P. van der Aalst. Understanding the occurrence of errors in process models based on metrics. In Meersman and Tari [254], pages 113–130.

[115] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[116] Dana Angluin. Inductive inference of formal languages from positive data. *Information and Control*, 45(2):117–135, 1980.

[117] Dana Angluin and Carl H. Smith. Inductive inference: Theory and methods. *ACM Comput. Surv.*, 15(3):237–269, 1983.

[118] Evelina Lamma, Paola Mello, Fabrizio Riguzzi, and Sergio Storari. Applying inductive logic programming to process mining. In Hendrik Blockeel, Jan Ramon, Jude W. Shavlik, and Prasad Tadepalli, editors, *ILP*, volume 4894 of *Lecture Notes in Computer Science*, pages 132–146. Springer, 2007.

[119] Marco Alberti, Federico Chesani, Marco Gavanelli, Evelina Lamma, Paola Mello, and Paolo Torroni. Verifiable agent interaction in abductive logic programming: The sciff framework. *ACM Trans. Comput. Log.*, 9(4), 2008.

[120] Federico Chesani, Evelina Lamma, Paola Mello, Marco Montali, Fabrizio Riguzzi, and Sergio Storari. Exploiting inductive logic programming techniques for declarative process mining. *T. Petri Nets and Other Models of Concurrency*, 2:278–295, 2009.

[121] Hendrik Blockeel and Luc De Raedt. Top-down induction of first-order logical decision trees. *Artif. Intell.*, 101(1-2):285–297, 1998.

[122] Stephen Muggleton. Inductive logic programming. In *ALT*, pages 42–62, 1990.

[123] Ross D. King. *Inductive logic programming: techniques and applications* by nada lavrac and saso dzeroski, ellis horwood, uk, 1993, pp 293, £39.95, isbn 0-13-457870-8. *Knowledge Eng. Review*, 9(3):311–312, 1994.

[124] Anne Rozinat, Ana Karla Alves de Medeiros, Christian W. Günther, A. J. M. M. Weijters, and Wil M. P. van der Aalst. The need for a process mining evaluation framework in research and practice. In ter Hofstede et al. [243], pages 84–89.

[125] H. Yang, B.F. van Dongen, A.H.M. ter Hofstede, M.T. Wynn, and J. Wang. Estimating Completeness of Event Logs. *BPM Center Report, 12-04-2012*, 2012.

[126] H. Yang, A.H.M. ter Hofstede, B.F. van Dongen, M.T. Wynn, and J. Wang. On Global Completeness of Event Logs. *BPM Center Report, 10-09-2012*, 2010.

[127] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3):249–260, 1995.

[128] Wil M. P. van der Aalst, H. T. de Beer, and Boudewijn F. van Dongen. Process mining and verification of properties: An approach based on temporal logic. In Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, Özalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, and Stefano Spaccapietra, editors, *OTM Conferences (1)*, volume 3760 of *Lecture Notes in Computer Science*, pages 130–147. Springer, 2005.

[129] Fahiem Bacchus and Froduald Kabanza. Using temporal logics to express search control knowledge for planning. *Artif. Intell.*, 116(1-2):123–191, 2000.

[130] Rajeev Alur and Thomas A. Henzinger. A really temporal logic. *J. ACM*, 41(1):181–204, 1994.

[131] Jan Chomicki. Efficient checking of temporal integrity constraints using bounded history encoding. *ACM Trans. Database Syst.*, 20(2):149–186, 1995.

[132] William W. Cohen. Fast effective rule induction. In Armand Prieditis and Stuart J. Russell, editors, *ICML*, pages 115–123. Morgan Kaufmann, 1995.

[133] Maja Pesic and Wil M. P. van der Aalst. A declarative approach for flexible business processes management. In Johann Eder and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 169–180. Springer, 2006.

[134] Maja Pesic, Helen Schonenberg, and Wil M. P. van der Aalst. Declare: Full support for loosely-structured processes. In *EDOC* [255], pages 287–300.

[135] Maja Pesic, M. H. Schonenberg, Natalia Sidorova, and Wil M. P. van der Aalst. Constraint-based workflow models: Change made easy. In Meersman and Tari [254], pages 77–94.

[136] Wil M. P. van der Aalst, Maja Pesic, and Helen Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - R&D*, 23(2):99–113, 2009.

[137] Federico Chesani, Paola Mello, Marco Montali, and Sergio Storari. Towards a decserflow declarative semantics based on computational logic. Technical Report DEIS-LIA-07-001, University of Bologna (Italy), January 2007. LIA Series no. 79.

[138] Edmund M. Clarke, Orna Grumberg, and A. Peled. *Model Checking*. MIT Press, 1999.

[139] M. Pesic and W. M. P. van der Aalst. Analyzing the resource perspective of workflow management systems: using a meta model and constraints. BETA working paper series 157, Eindhoven University of Technology, 2006.

[140] Wil M. P. van der Aalst and A. J. M. M. Weijters. Process mining: a research agenda. *Computers in Industry*, 53(3):231–244, 2004.

[141] Aldo Gangemi, Nicola Guarino, Claudio Masolo, Alessandro Oltramari, and Luc Schneider. Sweetening ontologies with dolce. In Asunción Gómez-Pérez and V. Richard Benjamins, editors, *EKAW*, volume 2473 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2002.

[142] Giancarlo Guizzardi and Gerd Wagner. Towards ontological foundations for agent modelling concepts using the unified fundational ontology (ufo). In Paolo Bresciani, Paolo Giorgini, Brian Henderson-Sellers, Graham Low, and Michael Winikoff, editors, *AOIS*, volume 3508 of *Lecture Notes in Computer Science*, pages 110–124. Springer, 2004.

[143] Giancarlo Guizzardi and Gerd Wagner. A unified foundational ontology and some applications of it in business modeling. In Janis Grundspenkis and Marite Kirikova, editors, *CAiSE Workshops (3)*, pages 129–143. Faculty of Computer Science and Information Technology, Riga Technical University, Riga, Latvia, 2004.

[144] Giancarlo Guizzardi, Gerd Wagner, Nicola Guarino, and Marten van Sinderen. An ontologically well-founded profile for uml conceptual models. In Anne Persson and Janis Stirna, editors, *CAiSE*, volume 3084 of *Lecture Notes in Computer Science*, pages 112–126. Springer, 2004.

[145] Guido L. Geerts and William E. McCarthy. An accounting object infrastructure for knowledge-based enterprise models. *IEEE Intelligent Systems*, 14(4):89–94, 1999.

[146] August-Wilhelm Scheer, Oliver Thomas, and Otmar Adam. Process modeling using event-driven process chains. In *Process-Aware Information Systems*. Wiley, 2005.

[147] David C. Luckham. *The power of events - an introduction to complex event processing in distributed enterprise systems.* ACM, 2005.

[148] Avigdor Gal and Ethan Hadar. Generic architecture of complex event processing systems. In Annika Hinze and Alejandro P. Buchmann, editors, *Principles and Applications of Distributed Event-Based Systems*, pages 1–18. IGI Global, 2010.

[149] Gian Piero Zarri. Representation and processing of complex events. In *AAAI Spring Symposium: Intelligent Event Processing*, pages 101–. AAAI, 2009.

[150] Di Wang, Elke A. Rundensteiner, and Richard T. Ellison. Active complex event processing over event streams. *PVLDB*, 4(10):634–645, 2011.

[151] Di Wang, Elke A. Rundensteiner, Richard T. Ellison, and Han Wang. Active complex event processing infrastructure: Monitoring and reacting to event streams. In Serge Abiteboul, Klemens Böhm, Christoph Koch, and Kian-Lee Tan, editors, *ICDE Workshops*, pages 249–254. IEEE, 2011.

[152] M. Shepperd, Q. Song, Z. Sun, and C. Mair. Data quality: Some comments on the nasa software defect data sets. *IEEE Transactions on Software Engineering*, Under review, 2012.

[153] Filip Caron, Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Bart Baesens. On the distinction between truthful, invisible, false and unobserved events an event existence classification framework and the impact on business process analytics related research areas. In *AMCIS*. Association for Information Systems, 2012.

[154] Jan Claes and Geert Poels. Process mining and the prom framework: An exploratory survey. In Rosa and Soffer [253], pages 187–198.

[155] Wil M. P. van der Aalst, Ana Karla A. de Medeiros, and A. J. M. M. Weijters. Genetic process mining. In Ciardo and Darondeau [246], pages 48–69.

[156] Ana Karla A. de Medeiros, A. J. M. M. Weijters, and Wil M. P. van der Aalst. Genetic process mining: A basic approach and its challenges. In Bussler and Haller [245], pages 203–215.

[157] Christian W. Günther. *Process Mining in Flexible Environments.* PhD thesis, TU Eindhoven, 2009.

[158] Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Conformance checking in the large: Partitioning and topology. In Daniel et al. [256], pages 130–145.

[159] Andrea Burattin and Alessandro Sperduti. Plg: A framework for the generation of business process models and their execution logs. In zur Muehlen and Su [252], pages 214–219.

[160] Wil M. P. van der Aalst, Boudewijn F. van Dongen, Joachim Herbst, Laura Maruster, Guido Schimm, and A. J. M. M. Weijters. Workflow mining: A survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.

[161] Object Management Group. Business Process Model and Notation (BPMN) Version 2.0. OMG Document – formal/2011-01-03, 2011.

[162] Object Management Group. Business Process Model and Notation (BPMN) Version 1.2. OMG Document – formal/2009-01-03, 2009.

[163] Remco M. Dijkman and Pieter Van Gorp. Bpmn 2.0 execution semantics formalized as graph rewrite rules. In Jan Mendling, Matthias Weidlich, and Mathias Weske, editors, *BPMN*, volume 67 of *Lecture Notes in Business Information Processing*, pages 16–30. Springer, 2010.

[164] Remco M. Dijkman, Marlon Dumas, and C. Ouyang. Formal semantics and automated analysis of BPmn process models. Technical report, Queensland University of Technology, 2007.

[165] Peter Y. H. Wong and Jeremy Gibbons. A process semantics for bpmn. In Shaoying Liu, T. S. E. Maibaum, and Keijiro Araki, editors, *ICFEM*, volume 5256 of *Lecture Notes in Computer Science*, pages 355–374. Springer, 2008.

[166] Vitus SW Lam. A precise execution semantics for bpmn. *IAENG International Journal of Computer Science*, 39(1), 2012.

[167] Michael zur Muehlen and Jan Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In Janis A. Bubenko Jr., John Krogstie, Oscar Pastor, Barbara Pernici, Colette Rolland, and Arne Sølvberg, editors, *Seminal Contributions to Information Systems Engineering*, pages 429–443. Springer, 2013.

[168] Jan Recker. Opportunities and constraints: the current struggle with bpmn. *Business Proc. Manag. Journal*, 16(1):181–201, 2010.

[169] Jan C Recker. BPMN modeling–who, where, how and why. *BPTrends*, 5(3):1–8, 2008.

[170] Michele Chinosi and Alberto Trombetta. Bpmn: An introduction to the standard. *Computer Standards & Interfaces*, 34(1):124–134, 2012.

[171] Marlon Dumas, Marcello La Rosa, Jan Mendling, and Hajo A. Reijers. *Fundamentals of Business Process Management.* Springer, 2013.

[172] Tomislav Rozman, Romana Vajde Horvat, and Ivan Rozman. Modeling the standard compliant software processes in the university environment. *Business Proc. Manag. Journal*, 14(1):53–64, 2008.

[173] Wil M. P. van der Aalst. On the representational bias in process mining. In Sumitra Reddy and Samir Tata, editors, *WETICE*, pages 2–7. IEEE Computer Society, 2011.

[174] Wil M. P. van der Aalst, Arya Adriansyah, Ana Karla Alves de Medeiros, Franco Arcieri, Thomas Baier, Tobias Blickle, R. P. Jagadeesh Chandra Bose, Peter van den Brand, Ronald Brandtjen, Joos C. A. M. Buijs, Andrea Burattin, Josep Carmona, Malú Castellanos, Jan Claes, Jonathan Cook, Nicola Costantini, Francisco Curbera, Ernesto Damiani, Massimiliano de Leoni, Pavlos Delias, Boudewijn F. van Dongen, Marlon Dumas, Schahram Dustdar, Dirk Fahland, Diogo R. Ferreira, Walid Gaaloul, Frank van Geffen, Sukriti Goel, Christian W. Günther, Antonella Guzzo, Paul Harmon, Arthur H. M. ter Hofstede, John Hoogland, Jon Espen Ingvaldsen, Koki Kato, Rudolf Kuhn, Akhil Kumar, Marcello La Rosa, Fabrizio Maria Maggi, Donato Malerba, R. S. Mans, Alberto Manuel, Martin McCreesh, Paola Mello, Jan Mendling, Marco Montali, Hamid R. Motahari Nezhad, Michael zur Muehlen, Jorge Munoz-Gama, Luigi Pontieri, Joel Ribeiro, Anne Rozinat, Hugo Seguel Pérez, Ricardo Seguel Pérez, Marcos Sepúlveda, Jim Sinur, Pnina Soffer, Minseok Song, Alessandro Sperduti, Giovanni Stilo, Casper Stoel, Keith D. Swenson, Maurizio Talamo, Wei Tan, Chris Turner, Jan Vanthienen, George Varvaressos, Eric Verbeek, Marc Verdonk, Roberto Vigo, Jianmin Wang, Barbara Weber, Matthias Weidlich, Ton Weijters, Lijie Wen, Michael Westergaard, and Moe Thandar Wynn. Process mining manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer, 2011.

[175] Wil M. P. van der Aalst, Hajo A. Reijers, and Minseok Song. Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593, 2005.

[176] Jian Pei, Jian Liu, Haixun Wang, Ke Wang, Philip S. Yu, and Jianyong Wang. Efficiently mining frequent closed partial orders. In *ICDM*, pages 753–756. IEEE Computer Society, 2005.

[177] Heikki Mannila, Hannu Toivonen, and A. Inkeri Verkamo. Discovery of frequent episodes in event sequences. *Data Min. Knowl. Discov.*, 1(3):259–289, 1997.

[178] Jian Pei, Haixun Wang, Jian Liu, Ke Wang, Jianyong Wang, and Philip S. Yu. Discovering frequent closed partial orders from strings. *IEEE Trans. Knowl. Data Eng.*, 18(11):1467–1481, 2006.

[179] Kuo-Yu Huang and Chia-Hui Chang. Efficient mining of frequent episodes from complex sequences. *Inf. Syst.*, 33(1):96–114, 2008.

[180] Wil M. P. van der Aalst. Decomposing petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013.

[181] Wil M. P. van der Aalst. Decomposing process mining problems using passages. In Haddad and Pomello [250], pages 72–91.

[182] Jorge Munoz-Gama, Josep Carmona, and Wil M. P. van der Aalst. Hierarchical conformance checking of process models based on event logs. In José Manuel Colom and Jörg Desel, editors, *Petri Nets*, volume 7927 of *Lecture Notes in Computer Science*, pages 291–310. Springer, 2013.

[183] Bokyoung Kang, Seung Kyung Lee, Yeong bin Min, Suk-Ho Kang, and Nam Wook Cho. Real-time process quality control for business activity monitoring. In Marina L. Gavrilova, Osvaldo Gervasi, David Taniar, Youngsong Mun, and Andrés Iglesias, editors, *ICCSA Workshops*, pages 237–242. IEEE Computer Society, 2009.

[184] Christian Janiesch, Martin Matzner, and Oliver Müller. Beyond process monitoring: a proof-of-concept of event-driven business activity management. *Business Proc. Manag. Journal*, 18(4):625–643, 2012.

[185] Artem Polyvyanyy, Jussi Vanhatalo, and Hagen Völzer. Simplified computation and generalization of the refined process structure tree. In Mario Bravetti and Tevfik Bultan, editors, *WS-FM*, volume 6551 of *Lecture Notes in Computer Science*, pages 25–41. Springer, 2010.

[186] Minseok Song, Christian W. Günther, and Wil M. P. van der Aalst. Trace clustering in process mining. In Danilo Ardagna, Massimo Mecella, and Jian Yang, editors, *Business Process Management Workshops*, volume 17 of *Lecture Notes in Business Information Processing*, pages 109–120. Springer, 2008.

[187] Diogo R. Ferreira, Marielba Zacarias, Miguel Malheiros, and Pedro Ferreira. Approaching process mining with sequence clustering: Experiments and findings. In Alonso et al. [247], pages 360–374.

[188] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Context aware trace clustering: Towards improving process mining results. In *SDM*, pages 401–412. SIAM, 2009.

[189] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Trace clustering based on conserved patterns: Towards achieving better process models. In Rinderle-Ma et al. [257], pages 170–181.

[190] Francesco Folino, Gianluigi Greco, Antonella Guzzo, and Luigi Pontieri. Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. *Data Knowl. Eng.*, 70(12):1005–1029, 2011.

[191] Jochen De Weerdt, Seppe K. L. M. vanden Broucke, Jan Vanthienen, and Bart Baesens. Active trace clustering for improved process discovery. *IEEE Trans. Knowl. Data Eng.*, 25(12):2708–2720, 2013.

[192] M. Song, H. Yang, Seyed Hossein Siadat, and M. Pechenizkiy. A comparative study of dimensionality reduction techniques to enhance trace clustering performances. *Expert Syst. Appl.*, 40(9):3722–3737, 2013.

[193] Chathura C. Ekanayake, Marlon Dumas, Luciano García-Bañuelos, and Marcello La Rosa. Slice, mine and

dice: Complexity-aware automated discovery of business process models. In Daniel et al. [256], pages 49–64.

[194] D. Martens and F. Provost. Explaining documents' classifications. Working paper CEDER-11-01, New York University - Stern School of Business, 2011.

[195] David Martens and Foster Provost. Explaining data-driven document classifications. *MISQ*, 38(1):73–99, 2014.

[196] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Abstractions in process mining: A taxonomy of patterns. In Dayal et al. [248], pages 159–175.

[197] Vladimir I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707–710, 1966.

[198] Igor V. Cadez, David Heckerman, Christopher Meek, Padhraic Smyth, and Steven White. Model-based clustering and visualization of navigation patterns on a web site. *Data Min. Knowl. Discov.*, 7(4):399–424, 2003.

[199] Gabriel M. Veiga and Diogo R. Ferreira. Understanding spaghetti models with sequence clustering for prom. In Rinderle-Ma et al. [257], pages 92–103.

[200] Richard E. Bellman. *Adaptive control processes - A guided tour.* Princeton University Press, 1961.

[201] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. On the surprising behavior of distance metrics in high dimensional space. *Database Theory—ICDT 2001*, pages 420–434, 2001.

[202] Rob J. van Glabbeek and Ursula Goltz. Refinement of actions and equivalence notions for concurrent systems. *Acta Inf.*, 37(4/5):229–327, 2001.

[203] Jan Hidders, Marlon Dumas, Wil M. P. van der Aalst, Arthur H. M. ter Hofstede, and Jan Verelst. When are two workflows the same? In Mike D. Atkinson and Frank K. H. A. Dehne, editors, *CATS*, volume 41 of *CRPIT*, pages 3–11. Australian Computer Society, 2005.

[204] Bartek Kiepuszewski, Arthur H. M. ter Hofstede, and Wil M. P. van der Aalst. Fundamentals of control flow in workflows. *Acta Inf.*, 39(3):143–209, 2003.

[205] Wil M. P. van der Aalst, Ana Karla A. de Medeiros, and A. J. M. M. Weijters. Process equivalence: Comparing two process models based on observed behavior. In Dustdar et al. [244], pages 129–144.

[206] Ana Karla Alves de Medeiros, Wil M. P. van der Aalst, and A. J. M. M. Weijters. Quantifying process equivalence based on observed behavior. *Data Knowl. Eng.*, 64(1):55–74, 2008.

[207] Remco M. Dijkman, Marlon Dumas, Boudewijn F. van Dongen, Reina Käärik, and Jan Mendling. Similarity of business process models: Metrics and evaluation. *Inf. Syst.*, 36(2):498–516, 2011.

[208] Remco M. Dijkman. Diagnosing differences between business process models. In Marlon Dumas, Manfred Reichert, and Ming-Chien Shan, editors, *BPM*, volume 5240 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 2008.

[209] Remco M. Dijkman. A classification of differences between similar businessprocesses. In *EDOC* [255], pages 37–50.

[210] Boudewijn F. van Dongen, Remco M. Dijkman, and Jan Mendling. Measuring similarity between business process models. In Zohra Bellahsene and Michel Léonard, editors, *CAiSE*, volume 5074 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2008.

[211] Haiping Zha, Jianmin Wang, Lijie Wen, Chaokun Wang, and Jiaguang Sun. A workflow net similarity measure based on transition adjacency relations. *Computers in Industry*, 61(5):463–471, 2010.

[212] J.R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993.

[213] Yi Liu, Taghi M. Khoshgoftaar, and Naeem Seliya. Evolutionary optimization of software quality modeling with multiple repositories. *IEEE Trans. Software Eng.*, 36(6):852–864, 2010.

[214] R. P. Jagadeesh Chandra Bose and Wil M. P. van der Aalst. Trace alignment in process mining: Opportunities for process diagnostics. In Hull et al. [251], pages 227–242.

[215] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Min. Knowl. Discov.*, 2(2):121–167, 1998.

[216] J. Ross Quinlan. Simplifying decision trees. *International Journal of Man-Machine Studies*, 27(3):221–234, 1987.

[217] Anne Rozinat and Wil M. P. van der Aalst. Conformance testing: Measuring the fit and appropriateness of event logs and process models. In Bussler and Haller [245], pages 163–176.

[218] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. On the Role of Fitness, Precision, Generalization and Simplicity in Process Discovery (to appear). In *20th International Conference on Cooperative Information Systems (CoopIS 2012)*, lncs, 2012.

[219] Massimiliano de Leoni, Fabrizio Maria Maggi, and Wil M. P. van der Aalst. Aligning event logs and declarative process models for conformance checking. In Alistair P. Barros, Avigdor Gal, and Ekkart

Kindler, editors, *BPM*, volume 7481 of *Lecture Notes in Computer Science*, pages 82–97. Springer, 2012.

[220] Jianmin Wang, Raymond K. Wong, Jianwei Ding, Qinlong Guo, and Lijie Wen. Efficient selection of process mining algorithms. *IEEE T. Services Computing*, 6(4):484–496, 2013.

[221] Jianmin Wang, Raymond K. Wong, Jianwei Ding, Qinlong Guo, and Lijie Wen. On recommendation of process mining algorithms. In Carole A. Goble, Peter P. Chen, and Jia Zhang, editors, *ICWS*, pages 311–318. IEEE, 2012.

[222] P. Weber, B. Bordbar, P. Tino, and B. Majeed. A framework for comparing process mining algorithms. In *GCC Conference and Exhibition (GCC), 2011 IEEE*, pages 625–628, 2011.

[223] Joel Ribeiro, Josep Carmona, Mustafa Mısır, and Michele Sebag. A recommender system for process discovery. In Shazia Sadiq, Pnina Soffer, and Hagen Völzer, editors, *Business Process Management*, volume 8659 of *Lecture Notes in Computer Science*, pages 67–83. Springer International Publishing, 2014.

[224] Kurt Jensen, Lars Michael Kristensen, and Lisa Wells. Coloured petri nets and cpn tools for modelling and validation of concurrent systems. *STTT*, 9(3-4):213–254, 2007.

[225] Kurt Jensen. An introduction to the theoretical aspects of coloured petri nets. In J. W. de Bakker, Willem P. de Roever, and Grzegorz Rozenberg, editors, *REX School/Symposium*, volume 803 of *Lecture Notes in Computer Science*, pages 230–272. Springer, 1993.

[226] A. K. Alves De Medeiros and C. W. Gunther. Process mining: Using cpn tools to create test logs for mining algorithms. In *Proceedings of the Sixth Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools*, pages 177–190, 2005.

[227] Michael Westergaard and Boudewijn van Dongen. Keyvaluesets: Event logs revisited, 2013.

[228] Petri Nets World. Complete overview of petri nets tools database.

[229] Nicholas J. Dingle, William J. Knottenbelt, and Tamas Suto. Pipe2: A tool for the performance evaluation of generalised stochastic petri nets. *SIGMETRICS Perform. Eval. Rev.*, 36(4):34–39, March 2009.

[230] Olaf Kummer, Frank Wienberg, Michael Duvigneau, Jörn Schumacher, Michael Köhler, Daniel Moldt, Heiko Rölke, and Rüdiger Valk. An extensible editor and simulation engine for petri nets: Renew. In Jordi Cortadella and Wolfgang Reisig, editors, *ICATPN*, volume 3099 of *Lecture Notes in Computer Science*, pages 484–493. Springer, 2004.

[231] Andreas Eckleder and Thomas Freytag. Woped 2.0 goes bpel 2.0. In Niels Lohmann and Karsten Wolf, editors, *AWPN*, volume 380 of *CEUR Workshop Proceedings*, pages 75–80. CEUR-WS.org, 2008.

[232] T. Freytag. WoPeD–Workflow Petri Net Designer. *University of Cooperative Education*, 2005.

[233] Kees M. van Hee, Olivia Oanea, Reinier Post, Lou J. Somers, and Jan Martijn E. M. van der Werf. Yasper: a tool for workflow modeling and analysis. In *ACSD*, pages 279–282. IEEE Computer Society, 2006.

[234] A. Rozinat, A.K. Alves de Medeiros, C.W. Gunther, A.J.M.M. Weijters, and W.M.P. van der Aalst. The need for a process mining evaluation framework in research and practice: position paper. In *Proceedings of the 2007 international conference on Business process management (BPM 2007)*, pages 84–89, 2007.

[235] Seppe K. L. M. vanden Broucke, Jochen De Weerdt, Bart Baesens, and Jan Vanthienen. Improved artificial negative event generation to enhance process event logs. In Jolita Ralyté, Xavier Franch, Sjaak Brinkkemper, and Stanislaw Wrycza, editors, *CAiSE*, volume 7328 of *Lecture Notes in Computer Science*, pages 254–269. Springer, 2012.

[236] Laura Sánchez-González, Félix García, Jan Mendling, Francisco Ruiz, and Mario Piattini. Prediction of business process model quality based on structural metrics. In Jeffrey Parsons, Motoshi Saeki, Peretz Shoval, Carson C. Woo, and Yair Wand, editors, *ER*, volume 6412 of *Lecture Notes in Computer Science*, pages 458–463. Springer, 2010.

[237] Janez Demsar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7:1–30, 2006.

[238] John W. Mauchly. Significance test for sphericity of a normal n-variate distribution. *The Annals of Mathematical Statistics*, 11(2):pp. 204–209, 1940.

[239] Samuel W Greenhouse and Seymour Geisser. On methods in the analysis of profile data. *Psychometrika*, 24(2):95–112, 1959.

[240] Huynh Huynh and Leonard S. Feldt. Estimation of the box correction for degrees of freedom from sample data in randomized block and split-plot designs. *Journal of Educational Statistics*, 1(1):pp. 69–82, 1976.

[241] Damián Pérez-Alfonso, Raykenler Yzquierdo-Herrera, and Manuel Lazo-Cortés. Recommendation of process discovery algorithms: a classification problem. In *Proceeding of: 5th Mexican Conference on Pattern Recognition (MCPR 2013), Research in Computing Science*, volume 61, 2013.

[242] Joel Ribeiro, Josep Carmona, Mustafa Misir, and Michele Sebag. A recommender system for process discovery. In *Business Process Management - 12th International Conference, BPM 2014*, 2014.

[243] Arthur H. M. ter Hofstede, Boualem Benatallah, and Hye-Young Paik, editors. *Business Process Management Workshops, BPM 2007 International Workshops, BPI, BPD, CBP, ProHealth, RefMod, semantics4ws, Brisbane,*

Australia, September 24, 2007, Revised Selected Papers, volume 4928 of *Lecture Notes in Computer Science*. Springer, 2008.

[244] Schahram Dustdar, José Luiz Fiadeiro, and Amit P. Sheth, editors. *Business Process Management, 4th International Conference, BPM 2006, Vienna, Austria, September 5-7, 2006, Proceedings*, volume 4102 of *Lecture Notes in Computer Science*. Springer, 2006.

[245] Christoph Bussler and Armin Haller, editors. *Business Process Management Workshops, BPM 2005 International Workshops, BPI, BPD, ENEI, BPRM, WSCOBPM, BPS, Nancy, France, September 5, 2005, Revised Selected Papers*, volume 3812, 2006.

[246] Gianfranco Ciardo and Philippe Darondeau, editors. *Applications and Theory of Petri Nets 2005, 26th International Conference, ICATPN 2005, Miami, USA, June 20-25, 2005, Proceedings*, volume 3536 of *Lecture Notes in Computer Science*. Springer, 2005.

[247] Gustavo Alonso, Peter Dadam, and Michael Rosemann, editors. *Business Process Management, 5th International Conference, BPM 2007, Brisbane, Australia, September 24-28, 2007, Proceedings*, volume 4714 of *Lecture Notes in Computer Science*. Springer, 2007.

[248] Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo A. Reijers, editors. *Business Process Management, 7th International Conference, BPM 2009, Ulm, Germany, September 8-10, 2009. Proceedings*, volume 5701 of *Lecture Notes in Computer Science*. Springer, 2009.

[249] *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2011, part of the IEEE Symposium Series on Computational Intelligence 2011, April 11-15, 2011, Paris, France*. IEEE, 2011.

[250] Serge Haddad and Lucia Pomello, editors. *Application and Theory of Petri Nets - 33rd International Conference, PETRI NETS 2012, Hamburg, Germany, June 25-29, 2012. Proceedings*, volume 7347 of *Lecture Notes in Computer Science*. Springer, 2012.

[251] Richard Hull, Jan Mendling, and Stefan Tai, editors. *Business Process Management - 8th International Conference, BPM 2010, Hoboken, NJ, USA, September 13-16, 2010. Proceedings*, volume 6336 of *Lecture Notes in Computer Science*. Springer, 2010.

[252] Michael zur Muehlen and Jianwen Su, editors. *Business Process Management Workshops - BPM 2010 International Workshops and Education Track, Hoboken, NJ, USA, September 13-15, 2010, Revised Selected Papers*, volume 66 of *Lecture Notes in Business Information Processing*. Springer, 2011.

[253] Marcello La Rosa and Pnina Soffer, editors. *Business Process Management Workshops - BPM 2012 International Workshops, Tallinn, Estonia, September 3, 2012. Revised Papers*, volume 132 of *Lecture Notes in Business Information Processing*. Springer, 2013.

[254] Robert Meersman and Zahir Tari, editors. *On the Move to Meaningful Internet Systems 2007: CoopIS, DOA, ODBASE, GADA, and IS, OTM Confederated International Conferences CoopIS, DOA, ODBASE, GADA, and IS 2007, Vilamoura, Portugal, November 25-30, 2007, Proceedings, Part I*, volume 4803 of *Lecture Notes in Computer Science*. Springer, 2007.

[255] *11th IEEE International Enterprise Distributed Object Computing Conference (EDOC 2007), 15-19 October 2007, Annapolis, Maryland, USA*. IEEE Computer Society, 2007.

[256] Florian Daniel, Jianmin Wang, and Barbara Weber, editors. *Business Process Management - 11th International Conference, BPM 2013, Beijing, China, August 26-30, 2013. Proceedings*, volume 8094 of *Lecture Notes in Computer Science*. Springer, 2013.

[257] Stefanie Rinderle-Ma, Shazia Wasim Sadiq, and Frank Leymann, editors. *Business Process Management Workshops, BPM 2009 International Workshops, Ulm, Germany, September 7, 2009. Revised Papers*, volume 43 of *Lecture Notes in Business Information Processing*. Springer, 2010.

Publication List

A recent and up-to-date list is available at:

http://www.seppe.net

Articles in Internationally Reviewed Academic Journals

- Caron, F., vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Advanced rule-based process analytics: applications for risk response decisions and management control activities. Expert Systems with Applications, accepted.
- Seret, A., vanden Broucke, S., Baesens, B., Vanthienen, J. (2014). A dynamic understanding of customer behavior processes based on clustering and sequence mining. Expert Systems with Applications, accepted.
- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2013). Determining process model precision and generalization with weighted artificial negative events. IEEE Transactions on Knowledge and Data Engineering, accepted.
- De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B. (2013). Active trace clustering for improved process discovery. IEEE Transactions on Knowledge and Data Engineering, 25 (12), 2708-2720.
- Van den Bulcke, T., Vanden Broucke, P., Van Hoof, V., Wouters, K., vanden Broucke, S., Smits, G., Smits, E., Proesmans, S., Van Genechten, T., Eyskens, F. (2011). Data mining methods for classification of Medium-Chain Acyl-CoA dehydrogenase deficiency (MCADD) using non-derivatized tandem MS neonatal screening data. Journal of Biomedical Informatics, 44 (2), 319-325.

Articles in International Scientific Conferences and Symposia

- vanden Broucke, S., Muñoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J. (2014). Event-based real-time decomposed conformance analysis, 18 pp. OnTheMove Federated Conferences & Workshops, CoopIS 2014 (CoopIS'14), Amantea, Calabria (Italy), 27-31 October 2014.

- Zhu, X., Zhu, G., vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Supporting Sustainable Business Processes through a Geospatial Context Extension. 2nd International Conference on Geo-Informatics in Resource Management & Sustainable Ecosystem (GRMSE'14). Michigan (USA), 3-5 October 2014.

- Xinwei Zhu, Guobin Zhu, vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Towards Location-Aware Process Modeling and Execution. Workshop on Data- & Artifact- centric BPM (DAB'14). Haifa (Israel), 7-11 September 2014, accepted.

- De Weerdt, J., vanden Broucke, S., Caron, F. (2014). Bidimensional Process Discovery for Mining BPMN Models. Workshop on Decision Mining & Modeling for Business Processes (DeMiMoP'14). Haifa (Israel), 7-11 September 2014, accepted.

- De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Explaining Clustered Process Instances, Business Process Management Conference 2014, Haifa (Israel), 7-11 September 2014, accepted.

- vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Declarative Process Discovery with Evolutionary Computing. 2014 IEEE Congress on Evolutionary Computation Proceedings: Vol. accepted. 2014 IEEE. Beijing (China), 6-11 July 2014.

- Low, W.Z., De Weerdt, J., ter Hostede, A.H.M., van der Aalst, W.M.P., vanden Broucke, S. (2014). Cost Optimisation of Business Process Execution. 2014 IEEE Congress on Evolutionary Computation Proceedings: Vol. accepted. 2014 IEEE. Beijing (China), 6-11 July 2014.

- vanden Broucke, S., Vanthienen, J., Baesens, B. (2013). Volvo IT Belgium VINST. Proceedings of the 3rd Business Process Intelligence Challenge co-located with 9th International Business Process Intelligence workshop (BPI 2013): Vol. 1052. Business Process Intelligence Challenge 2013 (BPIC 2013).

Beijing (China), 26 August 2013 (art.nr. 3). Aachen (Germany): RWTH Aachen University.

- vanden Broucke, S., Caron, F., Vanthienen, J., Baesens, B. (2013). Validating and enhancing declarative business process models based on allowed and non-occurring past behavior. Business Process Management Workshops. Workshop on Decision Mining & Modeling for Business Processes (DeMiMoP'13). Beijing (China), 26-30 August 2013.

- vanden Broucke, S., Delvaux, C., Freitas, J., Rogova, T., Vanthienen, J., Baesens, B. (2013). Uncovering the relationship between event log characteristics and process discovery techniques. Business Process Management Workshops. Workshop on Business Process Intelligence (BPI2013). Beijing (China), 26-30 August 2013.

- Seret, A., vanden Broucke, S., Baesens, B., Vanthienen, J. (2013). An exploratory approach for understanding customer behavior processes bases on clustering and sequence mining. Business Process Management Workshops. Workshop on Decision Mining & Modeling for Business Processes (DeMiMoP'13). Beijing (China), 26-30 August 2013.

- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2013). A comprehensive benchmarking framework (CoBeFra) for conformance analysis between procedural process models and event logs in ProM. Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining, CIDM 2013, part of the IEEE Symposium Series on Computational Intelligence 2013, SSCI 2013: vol. accepted. IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2013). Singapore, 16-19 April 2013.

- Caron, F., vanden Broucke, S., Vanthienen, J., Baesens, B. (2012). On the distinction between truthful, invisible, false and unobserved events. Proceedings of the 18th Americas Conference on Information Systems: Vol. e-pub. Americas Conference on Information Systems. Seattle, Washington (US), 9-12 August 2012 (art.nr. 24) Association for Information Systems.

- De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B. (2012). Leveraging process discovery with trace clustering and text mining for intelligent analysis of incident management processes. Evolutionary Computation (CEC), 2012 IEEE Congress on. Congress on Evolutionary Computation (CEC), 2012 IEEE. Brisbane (Australia), 10-15 June 2012 (pp. 1-8) IEEE computational intelligence society.

- CAiSE'12 vanden Broucke, S., De Weerdt, J., Baesens, B., Vanthienen, J. (2012). An improved artificial negative event generator to enhance process event logs. In Ralyt, J. (Ed.), Franch, X. (Ed.), Brinkkemper, S. (Ed.), Wrycza, S. (Ed.), Lecture Notes in Computer Science. International Conference on Advanced Information Systems Engineering (CAiSE'12). Gdansk (Poland), 25-29 June 2012 (pp. 254-269) Springer.

- ICIS JAIS 2012 Caron, F., vanden Broucke, S., Vanthienen, J., Baesens, B. (2012). On the distinction between truthful, invisible, false and unobserved events. Sprouts: Working Papers on Information Systems: vol. 12 (16). 11th JAIS Theory Development Workshop at ICIS 2012. Orlando, Florida, 16 December 2012.

Articles in Professionally Oriented Journals, Technical Reports, Book(s) (Chapters)

- Baesens, B., Backiel, A., vanden Broucke, S. (2015). Beginning Java: Object Oriented Programming for Business Applications. Wiley, forthcoming.

- vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Straightforward Petri net-based event log generation in ProM. FEB Research Report KBI_1417, 12 pp. Leuven (Belgium): KU Leuven - Faculty of Economics and Business.

- Baesens, B. (2014). Chapter: Business Process Analytics by vanden Broucke S. in Analytics in A Big Data World: The Essential Guide to Data Science and its Applications. Wiley, June 2014.

- vanden Broucke, S., Baesens, B., Lismont, J., Vanthienen, J. (2014). Sluit de lus: moderne technieken in Business Process Analytics. Informatie.

- Baesens, B., vanden Broucke, S., Dejaeger, K., Eerola, T., Goedhuys, L., Riis, M., Wehkamp, R. (2013). Cloudcomputing in analytics: de hype ontraadseld.

- vanden Broucke, S., Baesens, B., Vanthienen, J. (2013). Closing the loop: state of the art in business process analytics. Data Insight & Social BI: Executive Update, Cutter Consortium.

- vanden Broucke, S. (2013). What's in a name: Bitcoin, de digitale munteenheid. ECONnect (Q2), 40-41.

- Dejaeger, K., vanden Broucke, S., Eerola, T., Wehkamp, R., Goedhuys, L., Riis, M., Baesens, B. (2012). Beyond the hype: cloud computing in analytics. Finland: Techila Technologies.
- Dejaeger, K., vanden Broucke, S., Eerola, T., Wehkamp, R., Goedhuys, L., Riis, M., Baesens, B. (2012). Beyond the hype: cloud computing in analytics. Data Insight & Social BI: Executive Update, Cutter Consortium.
- vanden Broucke, S., Muñoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J. (2013). Event-based real-time decomposed conformance analysis, 21 pp: Polytechnic University of Catalonia, Department of Information Languages and Systems.
- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2013). On replaying process execution traces containing positive and negative events. FEB Research Report KBI_1311, 17 pp. Leuven (Belgium): KU Leuven - Faculty of Economics and Business.
- De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B. (2012). Leveraging process discovery with trace clustering and text mining for intelligent analysis of incident management processes. FEB Research Report KBI_1215, 9 pp. Leuven (Belgium): KU Leuven - Faculty of Economics and Business.
- vanden Broucke, S., De Weerdt, J., Vanthienen, J., Baesens, B. (2012). An improved process event log artificial negative event generator. FEB Research Report KBI_1216, 17 pp. Leuven (Belgium): KU Leuven - Faculty of Economics and Business.

Articles in Submission and in Preparation

- De Weerdt, J., vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Explaining Clustered Process Instances with Support Vector Machines, *in preparation for submission.*
- vanden Broucke, S., De Weerdt, J., Vanthienen J., Baesens, B. (2014). Fodina: a Robust and Flexible Heuristic Process Discovery Technique, *in preparation for submission.*
- vanden Broucke, S., Muñoz-Gama, J., Carmona, J., Baesens, B., Vanthienen, J. (2014). Event-based real-time decomposed conformance analysis, *in preparation for submission.*

- Xinwei Zhu, Guobin Zhu, vanden Broucke, S., Vanthienen, J., Baesens, B. (2014). Towards Location-Aware Process Modeling and Execution, *in submission (Decision Support Systems).*
- vanden Broucke, S., Caron, F., Lismont, J., Vanthienen, J., Baesens, B. (2014). On the Gap between Reality and Registration: A Business Event Analysis Classification Framework, *in submission (Journal of Information Technology & Management).*

Doctoral Dissertations from the Faculty of Business and Economics

Doctoral dissertations from the Faculty of Business and Economics, see:

http://www.kuleuven.be/doctoraatsverdediging/archief.htm