Mining Process Model Variants

Challenges, Techniques, Examples

Chen Li

MINING PROCESS MODEL VARIANTS: CHALLENGES, TECHNIQUES, EXAMPLES

Chen Li

Ph.D. dissertation committee:

Chairman and Secretary	
Prof. Dr. Ir. A.J. Mouthaan	University of Twente, The Netherlands
Promotors	
Prof. Dr. M. Reichert	University of Ulm, Germany
Prof. Dr. R. J. Wieringa	University of Twente, The Netherlands
Assistant promotor	
Dr. A. Wombacher	University of Twente, The Netherlands
Members	
Prof. Dr. Ir. W. M. P. van der Aalst	Eindhoven University of Technology, The Netherlands
Prof. Dr. P. M. G. Apers	University of Twente, The Netherlands
Prof. Dr. Ir. J. P. Katoen	University of Twente, the Netherlands
Prof. Dr. M. Weske	University of Potsdam, Germany
Prof. Dr. D. R. Ferreira	Technical University of Lisbon, Portugal

CTIT Ph.D. Thesis Series No. 10-178
Centre for Telematics and Information Technology
P.O. Box 217, 7500 AE
Enschede, the Netherlands

SIKS Dissertation Series No. 2010-47
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

The research reported in this thesis has been supportedby Nederlandse Organisatie voor Wetenschappelijk Onderzoek (NWO), under contract No. 612.066.512

Typeset with LaTeX. Printed and bound by Ipskamp Drukkers B.V.
Cover design by Yiquan Zhang.
ISSN: 1381-3617
ISBN: 978-90-365-3084-2
http://dx.doi.org/10.3990/1.9789036530842

MINING PROCESS MODEL VARIANTS: CHALLENGES, TECHNIQUES, EXAMPLES

DISSERTATION

to obtain
the degree of doctor at the University of Twente,
on the authority of the rector magnificus,
prof. dr. H. Brinksma,
on account of the decision of the graduation committee,
to be publicly defended
on Thursday 11 November, 2010 at 16:45

by

Chen Li

born on April 26, 1982
in Xi'an, Shaanxi, China

This dissertation has been approved by:

Prof. Dr. M. Reichert (promotor)
Prof. Dr. R. J. Wieringa (promotor)
Dr. A. Wombacher (assistant promotor)

For my parents,
who love and support me
in every possible way

Abstract

During the last years a new generation of process-aware information systems has emerged, which enables process model configurations at buildtime as well as process instance changes during runtime. Respective model adaptations result in large collections of process model variants that are derived from same process model, but slightly differ in structure. Generally, such model variants are expensive to maintain and configure. In this thesis, we present challenges, scenarios and algorithms for representing, comparing and mining such process model variants.

We first introduce the notion of process distance, which corresponds to the minimal number of high-level change operations needed for transforming one process model into another. In general, we presume that the shorter the average distance between a reference process model and related process variants is, the less changes are required for adapting the variants and the less efforts are needed for (future) process configuration. In this context, we present a method based on boolean algebra to compute the distance between two process models.

Starting with a collection of related process model variants, the major goal of this thesis is to discover a reference process model out of which these variants can be easily configured; i.e., a reference process model with minimal average distance to the variants. To achieve this goal we present two advanced algorithms which have their pros & cons, and that are applicable in different scenarios. Our clustering algorithm does not presume any knowledge about the original reference process model out of which the process model variants were configured. By only looking at the process model variants, this algorithm can quickly discover a reference process model in polynomial time, which allows us to scale up when solving real-world problems. The clustering algorithm further provides information on how well each part of the discovered reference model fits to the variants. Our heuristic algorithm, in turn, can take the original reference model into account as well. In particular, the user can control to what degree the discovered model differs from the original one. This way we can avoid spaghetti-like process models and additionally control how many changes we want to perform on the original reference model.

We systematically evaluate and compare the two algorithms based on simulations that comprise more than 7000 process models. Simulation results indicate good performance and make the differences between the two algorithms explicit. For example, the simulation results indicate that our clustering algorithm runs significantly faster than our heuristic algorithm. However, our heuristic algorithm can identify important changes at the beginning of the search and can discover better results than the clustering algorithm.

We successfully applied the two algorithms to cases from the automotive and the healthcare domain. During these case studies, the practical relevance and benefit of our work has become evident once more.

Overall, this Ph.D thesis will contribute to more intelligent information systems by learning from past adaptations and to an improved management of the variants by continuously evolving related reference process model.

Samenvatting

De afgelopen jaren is een nieuwe generatie procesbewuste informatiesystemen verschenen die zowel configuraties in de ontwerpfase als veranderingen in de procesinstanties mogelijk maakt. De respectievelijke modeladaptaties resulteren in grote verzamelingen procesmodelvarianten die van hetzelfde procesmodel zijn afgeleid maar die kleine structuurverschillen hebben. Over het algemeen zijn deze modelvarianten duur in onderhoud en configuratie. In dit proefschrift presenteren wij uitdagingen, scenario's en algoritmes voor representatie, vergelijking en mining van deze procesmodelvarianten.

We introduceren eerst het begrip procesafstand dat correspondeert met het minimale aantal high-level veranderingsoperaties dat nodig is voor het omzetten van het ene procesmodel in het andere. In het algemeen nemen we aan dat hoe kleiner de gemiddelde afstand tussen een referentie procesmodel en een gerelateerde procesvariant, hoe minder veranderingen nodig zijn voor het aanpassen van de varianten en hoe minder inspanning nodig is voor (toekomstige) procesconfiguratie. In deze context presenteren wij een methode die gebaseerd is op Booleaanse algebra voor het berekenen van de afstand tussen twee procesmodellen.

Startend met een verzameling gerelateerde procesmodelvarianten is het hoofddoel van dit proefschrift het vinden van een referentie procesmodel waaruit deze varianten gemakkelijk kunnen worden geconfigureerd; met andere woorden, een referentie procesmodel met een minimale afstand tot de varianten. Om dit doel te bereiken presenteren wij twee geavanceerde algoritmes die beiden voor- en nadelen hebben, en die toepasbaar zijn in verschillende scenario's. Ons clusteralgoritme veronderstelt geen kennis van het oorspronkelijke referentie procesmodel waaruit de procesmodelvarianten zijn geconfigureerd. Door alleen de procesmodelvarianten te beschouwen, is dit algoritme in staat snel een referentiemodel in polynomiale tijd te vinden, wat ons in staat stelt te schalen wanneer we real-world problemen oplossen. Daarnaast geeft het clusteralgoritme informatie over hoe goed ieder deel van het gevonden referentiemodel past met de varianten. Ons heuristiekalgoritme is echter in staat ook het oorspronkelijke referentiemodel in aanmerking te nemen. In het bijzonder kan de gebruiker bepalen hoeveel het gevonden model afwijkt van het origineel. Op deze manier voorkomen we spaghettiachtige procesmodellen en daarnaast houden we het aantal veranderingen dat we willen toepassen op het originele referentiemodel in de hand.

We evalueren en vergelijken systematisch de twee algoritmes gebaseerd op simulaties die uit meer dan 7000 procesmodellen bestaan. Simulatieresultaten wijzen op een goede prestatie en maken de verschillen tussen de twee algoritmes duidelijk. Bijvoorbeeld, de simulatieresultaten duiden aan dat ons clusteralgoritme significant sneller loopt dan ons heuristiekalgoritme. Echter, ons heuristiekalgoritme is in staat in het begin van het zoekproces belangrijke veranderingen te ontdekken en het is in staat betere resultaten te vinden dan het clusteralgoritme.

We hebben de twee algoritmes met succes toegepast op casussen uit de autoindustrie en de gezondheidszorg. Tijdens deze case studies zijn opnieuw de relevantie en voordelen van ons werk duidelijk geworden.

In zijn geheel draagt dit proefschrift bij aan meer intelligente informatiesystemen door te leren van vroegere aanpassingen en door een verbeterd management van de varianten door het continue evolueren van het gerelateerde referentie procesmodel.

Acknowledgement

After four years of struggles, I finally start to write the last part of my thesis. When all the names are popping up, I surely believe that I can never reach this far without your great supports.

My first thank goes to my promotor Prof. Manfred Reichert. Dear Manfred, you are truly a great supervisor! You have taken care of every little detail of my research, and supported me in every possible way. Although you went back to Germany at the first year of my Ph.D, I never actually feel that you are far away. From you, I understand what the German word "Doktorvater" really means, and I will be your student all the time. Besides, I would like to thank my other promotor Prof. Roel Wieringa. Thanks for all the supports and flexibilities you allowed me in the past years. I really appreciate your efforts to let me not feel alone in the group. At last, I would like to address my appreciation to my daily supervisor Dr. Andreas Wombacher. Dear Andreas, whenever we have a discussion, I always have the feeling that I am a goalkeeper who needs to defend all the penalty shoots. However, I am also confident that if I can succeed in defending all your questions, we will have a great paper waiting for us.

Besides my supervisors, I would like to thank all committee members for reading my thesis and providing useful feedbacks. Particularly, I would like to thank Prof. Wil van der Aalst for your detailed comments and insightful discussions. Though I need to admit that I have not done all your suggested experiments due to time constraints, the thesis has already been significantly improved by considering most of your comments. Thanks very much again for your time and efforts on improving my thesis.

I would also like to thank my colleagues from IS group for their supports in my research and daily life. Particularly I would like to thank Lianne for translating my abstract into Dutch; thank Suse for taking care of all administrative tasks and speaking Dutch with me; thank André for your feedback and interesting discussions; thank Virginia for guiding me through the graduation procedures and thank Zlatko for all the helps of my after-Ph.D career. In addition, I would like to thank Silja, Emmanuele, Jelena, Siv, Novica, Damiano and the rest of the group for all our interesting social events, coffee breaks, seminar talks ... which make our group a great, cozy and friendly place to work in.

Another group of my colleagues are a group of Germans; to be more precise, a group of Ulmers From University of Ulm. Though I stayed only about two months there, it was really a great time to be with you guys. I would like to thank Rüdiger for arranging my stay in Ulm and driving my around the city. I would also like to thank Vera (of course Oliver as well) for inviting me to your home, the great dinner and the very nice, but long and exhausting hiking trip ;) In addition, I would like to thank Matthias, Thao, Andreas, Jens for our dinner talks, paintball games and the really cold walks after lunch. Dominic and Bela, it was really nice to have you in my office, and Alena, thanks for your help on my case study. At last, I want to thank Ralph for your good words during my interview. Without you, I am not even sure whether I can start as a Ph.D or not.

Here, I would also like to thank two guys I haven't even met yet. Matthias Wettstein, thanks very much for your help on my statistic correlation analyses, which saved me from getting lost in statistical books. And ZHANG Yiquan, thanks for designing the cover of my thesis, which is so nice that I could never be able to do something like this. Of course, I need to thank Flavien and WAN Ying for introducing these nice guys to me and all your coordination works in between. I own you a great favor/dinner/beer... whatever you prefer :)

In the past two years, I spend a lot of my spare time in the Association of Chinese Students and Scholars in The Netherlands (ACSSNL). It was a great experience, during which I have made a lot of friends and learned a lot. Here, I would like to thank Mr. LUO Ping, Mr. ZHANG Xiaodong and MR. XIA Lei for your helps and guidances in the past years. YU Miao, MAO Ziqian, LIU Fangbin, RAO Xiangyu, XIANG Fei, LI Yuan, LI Jiayang and YANG Peng, it was great to work with you guys and and I believe our friendship will go beyond just co-workers in a same organization. Also, I would like to thank DING Ning for saving me for several times when our projects didn't really go well. LI Rui and JIU Qianqian, I also had a lot of fun when working with you in several projects. At last, I would also like to thank CHEN Songyue, ZHANG Xiao and XIE Yanbo for your helps when organizing events in Enschede.

Now come the friends. I would like to give my great appreciation first to JIN Mingliang and SHUI Lingling. Thanks for sharing your apartment with me and preparing the wonderful dinners. Lao JIN, it is a great pleasure to work with you in the student union, to play badminton with you, and to have holiday together with you. I also need to say sorry for your efforts on teaching me badminton. In addition, I would like to thank LI Nan & LIU Jun, ZHONG Zhicheng & HUANG Yanhui (and your daughter) for your hospitality. Thank ZHAO Yiping, SHENG Xiaoqin, HONG Liang, ZHANG Yang, YANG Di, SUN Chao for making my life joyful during my stay at Enschede. Also Joop, you are a great roommate. We had a lot of fun when living together and let's keep in touch. Kabir, thanks for being around and support me in various occasions. Without you, I think I will not even start my Ph.D here in The Netherlands.

For those who still haven't seen your names, trust me, my appreciations to you guys are not discounted. We both need to blame the publisher to who I have already agreed on the length of my thesis :)

At Last, I need to thank my parents. Without your support, I cannot even imagine of being who I am now. Your unconditional love is something I cannot even repay in my whole life. Also, Yuanyuan, knowing you was the greatest thing ever happened to me. Thanks for your love and support during my Ph.D, and I am looking forward to our futures.

Chen Li
October, 2010

Contents

Part I

Introduction

1 Motivation

1.1 Introduction

In today's dynamic business world success of an enterprise increasingly depends on its ability to react to changes in its environment in a quick, flexible and cost-effective way [128, 96, 38]. However, current off-the-shelf enterprise software (e.g., Enterprise Resource Planning systems, Hospital Information Systems, or Supply Chain Management Systems) has not fully met these needs yet [131]. Instead, it is deployed in different companies, domains, and countries, and therefore tends to be too generic and rigid. Usually, the introduction of enterprise software entails the problem of aligning business processes and IT [169, 76]. This causes huge customization efforts on the part software buyers that exceed the price for software licenses by factor five to ten [38, 76]. Software vendors, in turn, make endeavors to close this alignment gap [169, 186], and major progress has been achieved by shifting from function- to process-centered software design [196, 198, 225].

Along this trend a variety of process and service support paradigms (e.g., service orchestration [132], service choreography [132], and adaptive service [141, 28]) as well as corresponding specification languages (e.g., WS-BPEL [21], BPMN [22] and WSDL [233]) have emerged. *Process-aware information systems* (PAISs) offer promising perspectives in this respect, and a growing interest in aligning information systems in a process-oriented way can be observed [225]. In particular, PAISs allow to separate process logic and application code. This separation of concerns, in turn, increases maintainability and reduces cost of change [129, 213]. PAISs have become an integral part of enterprise computing and are used to support business processes at an operational level [225].

With the increasing adoption of PAISs, large *process model repositories* have emerged. Typically, the models in such repositories are re-aligned to real-world events and demands through adaptation on a day-to-day basis. In large companies, such process repositories can easily contain several thousands of process models [164]. Such sheer numbers give rise to several quality issues. Over time new process models emerge, existing ones need to be adapted to changing requirements, and new process model variants are created to align processes to a

particular context (e.g., country or product-specific regulations).

To ease the maintenance and evolution of such large process repositories, different approaches for flexible and adaptive processes exist [141, 154, 165]. Besides their use for behavior-preserving model refactorings [210], structural process adaptations (i.e., to add, delete or move process steps) are needed for customizing a reference process model to a particular context at buildtime (- such a reference model embodies the basic goal or general idea of a certain process type or can be looked as a reference for various purposes [163, 165]) [66, 165]. Furthermore, structural model adaptations may become necessary for adapting single process instances during runtime in order to deal with exceptional situations and changing needs [141, 211]. Altogether, the ability to effectively deal with process change has been identified as one of the most fundamental success factors for any PAISs [124, 133, 215, 211, 213, 209].

As example consider medical guidelines that exist for treating patients with a particular disease [96]. First, such process-centered guideline needs to be customized to fit to the particular healthcare environment in which it is applied. Second, additional adaptations might become necessary on-demand when applying it to a particular patient and his case [96]. Generally, in domains like healthcare [96, 4, 137] or automotive engineering [122, 127], no user would accept a PAIS if rigidity came with it [150].

Overall, the discussed ability to adapt process models at different levels, will lead to collections of process model variants (i.e., model configurations [69]) and to large process repositories. Thereby, this thesis will consider such process variant collections at buildtime and process variant collection that results from runtime adaptation.

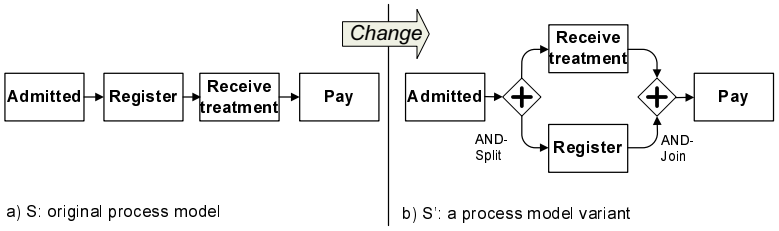


Figure 1.1: Original process model S and derived process variant S'

Fig. 1.1 depicts a very simple example. The left hand side shows a high-level view on a patient treatment process as it is normally executed: a patient is `admitted` to a hospital, where he first `registers`, then `receives treatment`, and finally `pays`. In emergency situations, however, it might become necessary to deviate from this model, e.g., by first starting treatment of the patient and allowing him to register later during treatment. To capture this behavior in the model of the respective process instance, we need to move activity `receive treatment` from its current position to a position parallel to activity `register`. This leads to an instance-specific process model variant S' as shown on the right

hand side of Fig. 1.1.

Generally, a large number of process model variants (*process variants* for short) derived from the same original process model might exist [107, 217, 128, 164]. For example, in the healthcare domain, we identified more than 90 process variants of a particular medical order handling procedure (see Chapter 9).

The problem looks even more challenging when additionally considering the customization and configuration at the different levels described above. Fig. 1.2 visualizes this problem: a domain-specific process reference model is configured into site-specific reference models for different customers. These site-specific reference models, in turn, may be further customized at *runtime* in order to cope with case- and instance specific requirements (cf. Fig 1.1) [69]. It is not difficult to imagine the complexity to be mastered if a large and deep tree of such process variants is built up in a process repository [107].

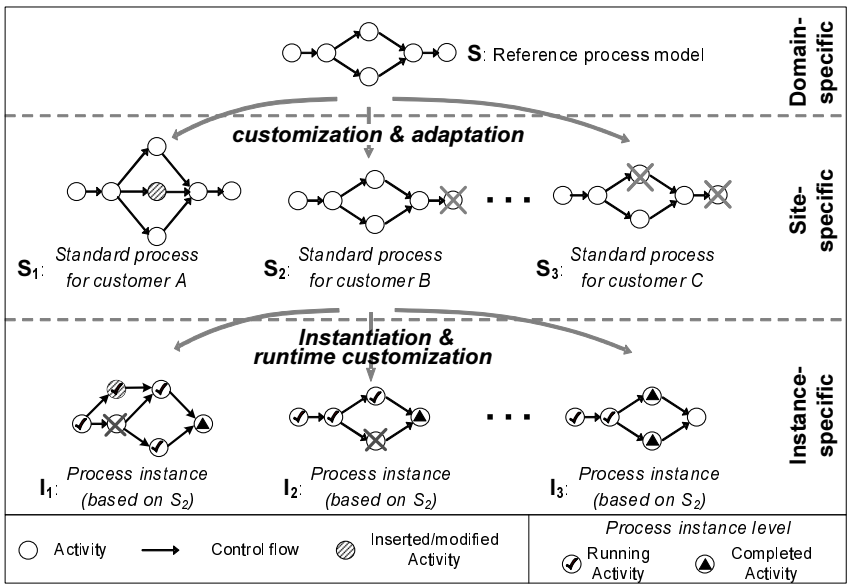


Figure 1.2: Domain-, site- and instance-specific process configurations

In most approaches supporting the adaptation and configuration of process models, each resulting process variant has to be maintained by its own [66], and even simple changes within a domain or organization (e.g. due to new laws or re-engineering efforts) might require manual re-editing of a large number of (logically related) process variants [215, 211]. Over time this leads to a degeneration and divergence of the respective process models [131], which aggravates maintenance significantly and which makes expensive refactorings indispensable.

1.2 Problem Statement

Though considerable efforts have been made to ease process configuration and process adaptation [217, 66, 141, 165], a notable research gap exists concerning quality assurance in large process repositories, and process variants collections respectively. Most existing approaches have not utilized information about process variant collections in the process repository yet [213]. Fig. 1.3 describes the overall goal of our research. We want to learn from related process variants in order to discover a (new) *reference process model* covering the given variant collection best. By adopting the discovered model in the PAIS, need for future process adaptations and costs for change will decrease. Generally, finding such reference model is by far not trivial when considering control flow patterns like sequence, parallel branching, conditional branching, and loops.

Fig. 1.3 further differentiates between two scenarios. In the *first scenario* there is only a collection of related process variants, but no knowledge about the original reference process model these variants were derived from. Here we want to discover a reference process model by "merging" common or frequent parts of these variants into one model. When adopting the discovered model as reference process model, we expect future process configurations to be reduced.

In the *second scenario* the process variants have been derived by configuring a known reference process model. When mining the new reference process model without considering the current one, however, we might be confronted with significant structural differences between old and new reference model. In most cases, "dramatic" changes of the current reference process model might be not preferred due to high implementation costs or for social reasons [71, 161, 173]. Therefore, process engineers should have the flexibility to control to what degree they want to maximally modify the original reference model such that the resulting model fits better to the given variant collection. Consequently, closeness of the new

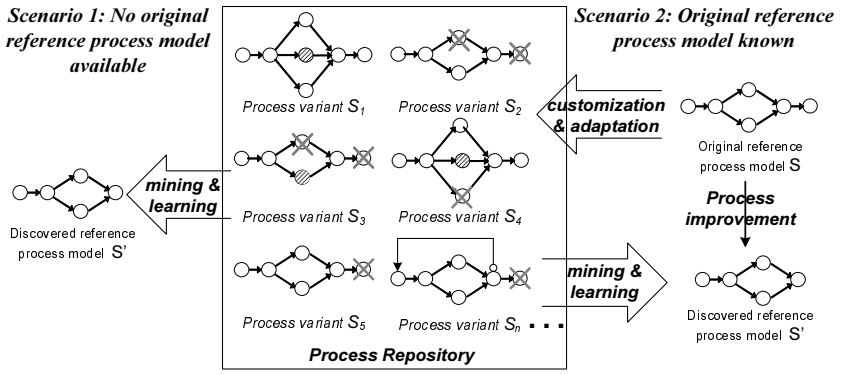


Figure 1.3: Different scenarios for discovering reference process models

reference model to the old one and closeness of this model to the variant models act as "counterforces". Ideally, the described flexibility also enables designers to consider only the most relevant configurations and adaptations respectively when evolving the reference process model.

This thesis deals with a number of research questions regarding the above mentioned scenarios. Basically, we distinguish between *knowledge problems* (KP) and *design problems* (DP) [227].[1] These research questions guide the research presented in this thesis:

- **Research Question 1 (KP)** What are fundamental challenges in mining process model variants? Are existing mining techniques suitable for realizing the goal of reducing process configuration efforts?
- **Research Question 2 (DP)** How shall we measure the distance between two process models such that this measure reflects minimal efforts for process model configurations?
- **Research Question 3 (DP)** Given a collection of process variants, how can we discover a reference process model in such a way that average distance between it and the process variants becomes minimal?
- **Research Question 4 (DP)** Given the original reference process model and a collection of related process variants derived from it, how can we derive a new reference process model that fits "better" to these variants? And how can we control the evolution of the reference process model in this context, i.e., how can we enable process engineers to control to what degree the new reference model may "differ" from the original one and how "close" it is to the given collection of process variants.
- **Research Question 5 (KP)** What are characteristic properties of the solution approaches we propose for supporting the different scenarios? Under which circumstances is the one approach better suited than the other?

When considering *Research Question 1*, we first try to identify the goals, scientific challenges and technical issues that emerge when mining process model variants. Based on this, we evaluate whether or not current approaches can help us in achieving the defined goals. Regarding *Research Question 2* we measure closeness (or distance) between a reference process model and a process variant in terms of the number of high-level change operations (e.g., to insert, delete or move activities) needed to transform the reference process model into the

[1] A *knowledge problem* is a difference between what we know about the world and what we would like to know [226]. Knowledge problems can be solved by asking others, by searching the literature, or by doing research. Knowledge problems have stakeholders, namely the people who would like to acquire the desired knowledge. Research problems typically are knowledge problems in which we search for true propositions. *Design problems*, in turn, are engineering problems, in which we search for an improvement of the world with respect to some goals. The evaluation criteria for answers to both kinds of problems are quite different: *truth* in the case of research problems, *goal achievement* in case of design problems.

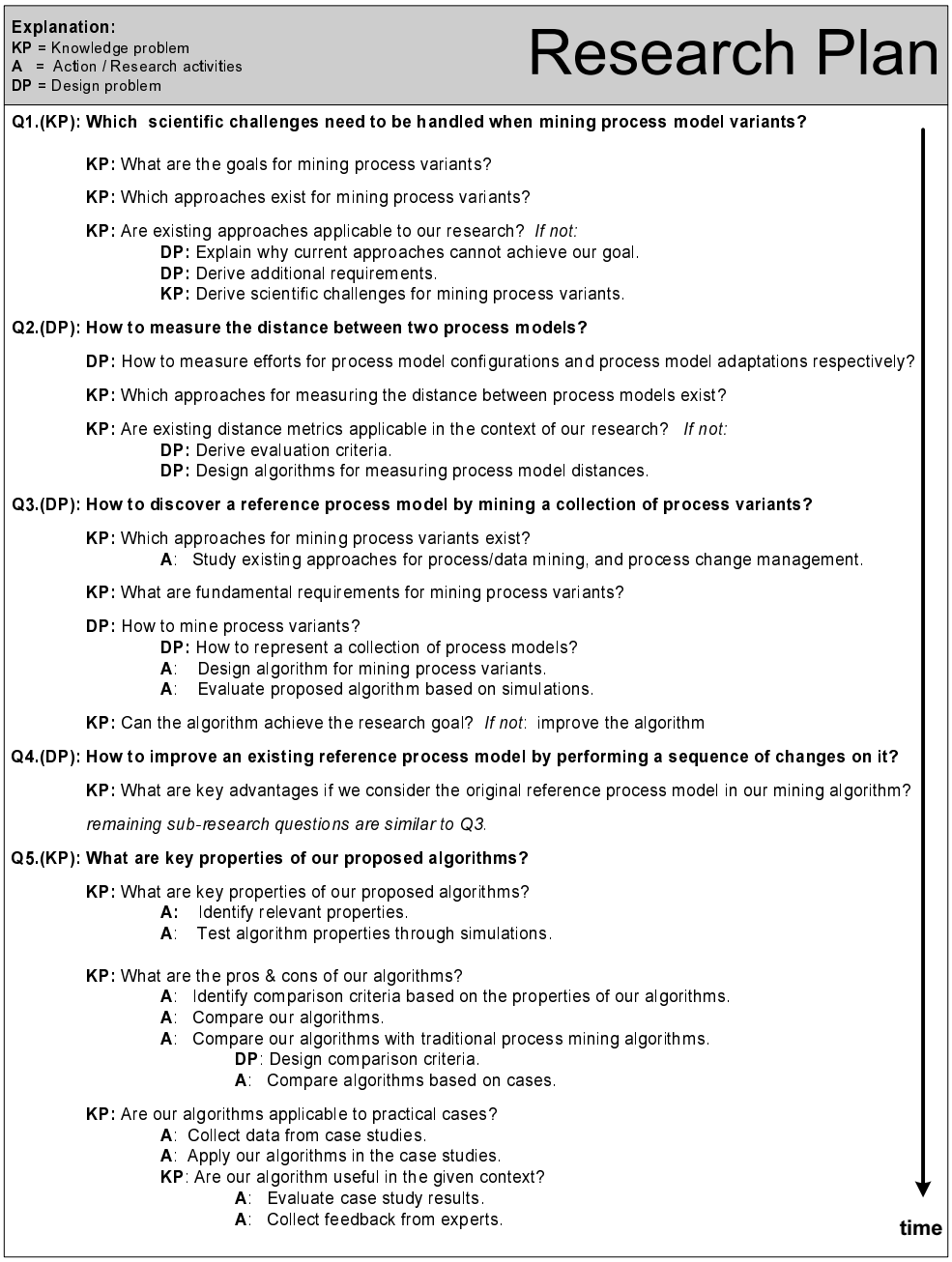


Figure 1.4: Research Plan

respective variant. Clearly, the shorter this distance is the less the efforts for configuring this variant will be.

Research Question 3 relates to the challenges discussed in connection with Scenario 1; i.e., we want to "merge" the most common and relevant parts of the different variants together in a reference process model. By adopting this reference process model in the PAIS, we can expect less configuration effort in future.

We try to handle the challenges set out by *Scenario 2* by answering *Research Question 4*. Here, we discover a new reference model by constructing a sequence of change operations to be applied the original one. Thereby, process engineers have the flexibility to control the similarity between the original reference model and the newly discovered one; i.e., to specify how many change operations shall be maximally applied to the old reference model when discovering the new one. As major benefits, we can control the efforts for updating the reference process model, and thus we can avoid Spaghetti-like model structures, which is a common challenge in the field of process mining [168, 39]. Clearly, the most relevant changes (i.e., the changes which significantly contribute to reduce the average distance between the newly discovered reference model and the variants) should be considered first and the less relevant ones last.

By working on *Research Question 5*, we finally evaluate and compare the properties of the solution approaches we propose for the support of the two scenarios. For example, we can measure execution time, quality parameters (e.g., distance reduction), scalability, and statistical properties of our solution approaches under different circumstances. The evaluation and comparison are performed based on *simulations* and studies of *real-world cases*.

Corresponding sub-research questions and a research plan are given in Fig. 1.4. Section 1.5 relates each research question to one of the following chapters.

1.3 Research Methodology

Like most software research & development projects [88, 13], our research comprises four phases: (1) *problem analysis*, (2) *solution Design*, (3) *implementation* and (4) *evaluation*. The research steps and the actions applied in each step are depicted in Fig. 1.5.

We start with a comprehensive literature study in *Phase 1*. Thereby, we focus on topics like process-aware information systems, (dynamic) process changes, process configuration, and process mining. Particularly, we are interested in techniques for managing and mining process model variants.

Based on the results of our literature study, we elicit requirements for mining process variants and start to design solutions (*Phase 2*). In this phase, we study literature relating to data mining, process mining, machine learning and reasoning techniques. Inspired by respective approaches, we design our algorithms for mining process variant collections.

The developed algorithms are implemented as stand-alone tool based on Java

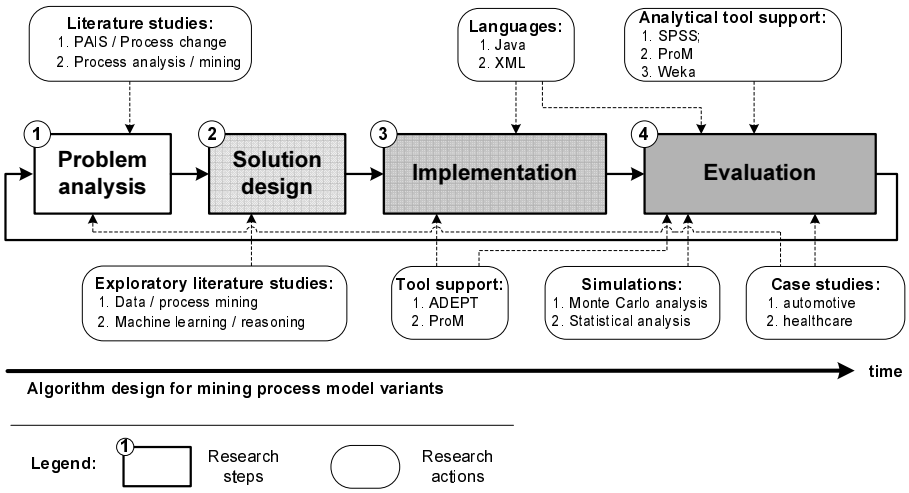


Figure 1.5: Research Methodology

(*Phase 3*). This tool is connected (via XML) with the ADEPT process management system [141] and the ProM process mining tool [201], which constitute two of the most popular tools for enabling (dynamic) process changes and process mining respectively.

The implemented algorithms are evaluated based on simulations and real-world cases (*Phase 4*). In this phase, we apply our algorithms in two case studies from automotive and healthcare domain. In addition, we perform various statistical analyses (mainly based on SPSS [179]) and data analyses (mainly based on Weka [228]) to evaluate the performance of our algorithms. Note that our research method constitutes an iterative approach. For example, the insights we obtain from our simulations and case studies contribute to further improvement of our algorithms, which trigger a new iteration of the four steps.

1.4 Contribution

In this thesis we develop a heuristic as well as a clustering algorithm for learning from previous process model adaptations and for discovering a reference model out of which the relating variants can be configured with minimum efforts. Each algorithm has its pros and cons. We systematically evaluate and compare the two algorithms through simulation and by applying them to real-world scenarios. In detail, the contributions of this thesis can be summarized as follows:

- We identify the research goals, scientific challenges and technical issues that emerge when mining process model variants. Through qualitative as well as quantitative analyses, we explain why current approaches cannot help us achieving the identified goals.

- We introduce the notion of order matrix for representing block-structured process models. An order matrix captures (transitive) order relations between pairs of process activities. Using order matrices, we can facilitate dynamic process changes and perform advanced process model analyses as required in the context of our mining approach.
- We present a method using Boolean algebra optimization to measure the distance between two process models. In this context, distance measures the minimal number of high-level change operations needed to transform one model into another. Thus it reflects overall efforts for process configuration.
- We design a clustering algorithm which can discover a reference process model by mining a collection of process variants. By adopting this reference process model within the PAIS, we expect lower process configuration efforts in future.
- We design a heuristic algorithm which can improve an existing reference process model by mining a process variant collection. When compared with our clustering algorithm, the heuristic algorithm can additionally control the difference between the discovered reference model and the original one; i.e., it can differentiate between important changes and trivial ones.
- We additionally evaluate these two algorithms based on simulations and case studies. Overall, simulation results indicate good performance of our algorithms, and case study results underline the high practical relevance of our work.

1.5 Outline of the Thesis

The remainder of this thesis is divided into four parts - *introduction, solution, validation* and *summary*. The outline of the thesis is depicted in Fig. 1.6.

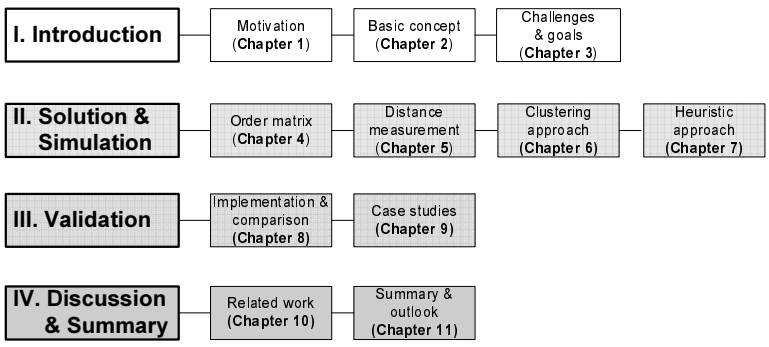


Figure 1.6: Outline of the Thesis

Part I comprises introductory chapters. More specifically, Chapter 1 first motivates the need for mining process variants and Chapter 2 introduces basic concepts needed for understanding this thesis. Chapter 3 then discusses goals and scientific challenges for mining process variants and shortly explains why current process mining techniques are unable to achieve the defined goal.

Part II provides novel techniques for representing, comparing and mining process variants. Chapter 4 first introduces the notion of *order matrix* which represents a block-structured process model by capturing the transitive order relations for pairs of activities. Chapter 5 then introduces an algorithm to evaluate the *distance* between two block-structured process models. In this context, we define distance as minimal number of high-level change operations (e.g., delete, insert, move activities) needed to transform one model into another. This, in turn, can reflect the efforts for process model configuration. Chapters 6 and 7 introduce a clustering and a heuristic algorithm respectively, which contribute to achieve the goals of this thesis for different settings, i.e., they can discover a reference process model out of which the process variants can be configured with lowest efforts. In addition, we systematically test the properties of our algorithms by comprehensive simulations based on several thousand process models.

Part III of this thesis evaluates the algorithms we develop in *Part II*. In Chapter 8, we first present a high-level architecture of our implemented tools and compare the developed algorithms qualitatively and quantitatively. Chapter 9 then presents two case studies where we successfully apply our algorithms to cases from the automotive industry and the healthcare domain.

Finally, *Part IV* discusses related work in Chapter 10 and summarizes main contributions of the thesis in Chapter 11. Chapter 11 also provides an outlook on future research directions.

Fig. 1.7 illustrates which research questions are addressed in which chapters. Since Chapter 2 introduces basic concepts, it relates to all research questions.

Research Questions \ Chapters	Chapter 1	Chapter 2	Chapter 3	Chapter 4	Chapter 5	Chapter 6	Chapter 7	Chapter 8	Chapter 9	Chapter 10	Chapter 11
Research question 1		✓	✓					✓		✓	
Research question 2		✓		✓	✓						
Research question 3		✓		✓		✓					
Research question 4		✓		✓			✓				
Research question 5		✓				✓	✓	✓	✓		

Figure 1.7: Research questions along the different chapters

2
Basic Concepts and Notions

This chapter introduces basic concepts needed for understanding this thesis. We first introduce the notion of Process-aware information system (PAIS) and related concepts in Section 2.1. Then we discuss basic concepts of adaptive processes in Section 2.2. In Section 2.3 we introduce the ADEPT process management technology, which provides advanced features for enabling correct process adaptations at different levels. ADEPT further addresses a broad spectrum of topics related to business process management (e.g., process modeling, process compliance, and process schema evolution). We present the full process lifecycle in a PAIS in Section 2.4. Finally, we briefly introduce process mining techniques in Section 2.5.

2.1 Process-aware Information Systems

This section introduces basic concepts of a PAIS: *process model, process structure tree, process instance* and *execution log.*

2.1.1 Process Model

A *process management system* (PrMS) provides generic process support functions and allows for separating process logic and application code. For this purpose, the process logic has to be explicitly defined based on the modeling patterns provided by the underlying a *process meta model.* At runtime the PrMS then orchestrates the processes according to the defined logic. For each business process to be supported, a *process type*, represented by a **process model** S, has to be defined.

In this thesis, a single process model is represented as a directed graph, which comprises a set of nodes - either representing *process steps* (i.e., *activities*) or control connectors (e.g, *And-/Xor-Split*) – and a set of *control edges* between them. The latter specify *precedence* as well as *loop backward relations.* Furthermore, we presume that process models are *block-structured* (see below).

Fig. 2.1 depicts an example of such a block-structured process model. Nodes are represented as rectangles while precedence and loop backwards relations are expressed as directed edges of different type. Each process model contains a

unique start and a unique end node.[1] For control flow modeling the following patterns are available: Sequence, AND-split, AND-join, XOR-split, XOR-join, and Loop [193]. These patterns constitute the core of any process specification language and cover most of the process models we can find in practice [236, 113]. Further, they can be easily mapped to other process execution languages like WS-BPEL (Web Service Business Process Execution Language) as well as to formal languages like Petri Nets [21, 196]. Based on these patterns we are also able to compose more complex process structures if required (e.g., in principle, an OR-split can be mapped to AND- and XOR-splits [117]). Finally, by only using these basic process patterns, we obtain better understandable and less erroneous models [116].

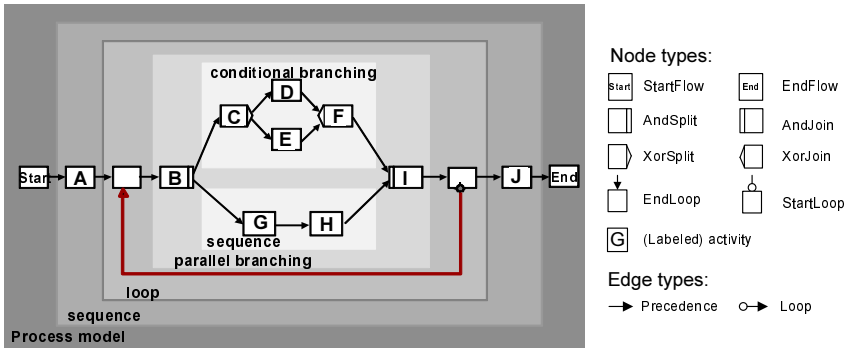


Figure 2.1: Block-structured process model

Each node n of a process model S may have a label $l(n)$. Such labeled nodes constitute the **activities** of S (e.g., activities `A` and `B` in Fig. 2.1). However, not all nodes must be labeled. We denote nodes without associated label as silent activities. These have no associated actions and only exist for control flow purpose (e.g., the StartLoop and EndLoop nodes in Fig. 2.1) [100]. In the context our research, for labeled activities we assume that their label is unique. Regarding our example from Fig. 2.1, S contains 10 normal activities and 4 silent ones.

As aforementioned, we further presume that a process model S is *block-structured* (cf. Fig. 2.1); i.e., activities, sequences, branchings, and loops constitute blocks with unique *start* and *end* nodes (or the *start* and end of such block become clear from the model)[2] [140, 148, 204, 84]. These blocks may be nested, but must not overlap; i.e., their nesting must be regular (cf. Appendix A.1 for a formal description of the properties of block-structured models) [141, 84, 140, 204]. Generally a block can be a single activity, a sequence, a parallel branching, a conditional branching, or S itself (cf. Appendix A.1). In

[1] For the sake of readability, we omit the start and end node of a process model if its start and end are clear. This applies to most process models in later Chapters.

[2] For example, in Fig. 2.1 the conditional branching block that comprises activities `C,D,E` and `F` constitutes a block.

Fig. 2.1, the grey areas show selected blocks of process model S, and the different grey levels indicate their nesting. In principle, we can consider a block itself as a block-structured process model. In the following, we represent each block as set of activities since the block-structure itself can be derived from the given process model S; e.g., block {G,H} corresponds to the sequence structure in S. Similarly, {A}, {C,D,E,F}, {B,C,D,E,F,G,H,I}, and {A,B,C,D,E,F,G,H,I,J} describe selected blocks contained in S (cf. Fig. 2.1).[3]

The concept of block-structuring has been known from block-structured programming languages for a long time [45], and can be (partly) found in process specification language like WS-BPEL and XLANG as well. Furthermore, process management systems like AristaFlow BPM Suite [37] and CAKE2 [119] emerged, which have been applied to a variety of processes from different domains and whose process modeling language is block-structured. In the context of our framework, the block structuring of the process meta model was motivated by three aspects:[4]

- First, when compared with non-block-structured process models, block-structured models are easier understandable for users and have less chances of containing errors [151, 114, 115, 116, 32]. This will be particularly important if users need to adapt process models during run-time.
- Second, block-structuring makes it possible to restrict the area in the graph to be analyzed in the context of changes. This, in turn, allows for quick abstractions and speeds up required analyses [141].
- Third, block-structure simplifies structural adaptations of the process schema and enables a variety of high-level change patterns [211].

If a process model is not block-structured, in many cases we can transform it into a block-structured representation [204, 116, 84, 140]. For example, we analyzed 214 process models from different domains, which were expressed in different languages (e.g., Event Process Chains, UML Activity Diagrams). More than 95% of them were block-structured or could be transformed into a block-structured representation [182, 135]. Despite the fact that there exist process models which are not block-structured and which cannot be transformed into block-structure, we consider our mining algorithms for block-structured process variant models as being highly relevant. Formally, we define **block-structured process model** as follows:

Definition 1 (Block-structured Process Model) *A tuple* $S = (A, E, AT, ET, l)$ *is called a block-structured process model if the following holds:*

[3] Here, {B,C,D,E,F,G,H,I} only refers to the parallel branching. It will become clear in Chapter 4 how we represent the loop block.

[4] This research has been conducted in the context of the ADEPT framework (cf. Section 2.3). Note that the core ADEPT meta model also contains elements that increase expressiveness when compared to pure block-structured models. Examples include synchronization and failure edges (see [141] for details).

- *A is a set of* `nodes` *and AT assigns to each node $a \in A$ a* `node type` *$AT(a) \in \{$`StartFlow, EndFlow, Normal, AndSplit, AndJoin, XorSplit, XorJoin, StartLoop, EndLoop`$\}$*
- *$E \subseteq A \times A$ is a set of* `directed edges` *and ET assigns to each edge $e \in E$ an* `edge type` *$ET(e) \in \{$`Precedence,Loop`$\}$. Further, $n \prec m :\Leftrightarrow n$ directly or indirectly precedes m when only considering* `precedence` *edges.*
- *Let L be a set of activity labels. $l : A \rightarrow L$ is a partial* `labeling function` *which assigns a label $l(a) \in L$ to a node $a \in A$.*
- *S has the block-structure properties as described above (for a formal and precise description see Appendix A.1).*

2.1.2 Process Structure Tree

Transforming a block-structured model into a tree representation has its roots in *structured programming* and compiler theory [3]. Such transformation can be applied, for example, when analyzing block-structured languages like XML or WS-BPEL. In the context of this thesis, we apply the approach presented in [204], which can transform a block-structured process model S into a *refined process structure tree* in linear time. Such refined process structure tree constitutes a unique representation of a block-structured process model. In the following, we denote it as process structure tree for short.

Definition 2 (Process Structure Tree) *A tuple $T = (N, C, CT, E, l)$ is called a process structure tree if the following holds:*

- *N is a set of* `nodes`*.*
- *C is a set of* `connectors` *and CT assigns to each connector $c \in C$ a* `connector type` *$CT(c) \in \{$`Seq, AND, XOR, Loop`$\}$.*
- *$E \subseteq (C \times C) \bigcup (C \times N)$ is a set of directed edges.*
- *Let L be a set of activity labels, $l : N \rightarrow L$ is a partial* `labeling function` *which assigns a label $l(n)$ to a node $n \in N$.*

A process structure tree consists of a set of nodes, a set of connectors, and a set of directed edges linking them. The labeling function l assigns labels to nodes in a similar way as described in Definition 1. Fig. 2.2 shows a block-structured process model S and its corresponding process structure tree T. In such a tree, nodes (represented as rectangles) correspond to activities while connectors (represented as ellipses) represent their relations based on process patterns like Sequence, AND-block, XOR-block, and Loop [193]. The precedence relations (expressed by connector `Seq`) are parsed from left to right; e.g., activity `A` precedes the Loop block (i.e., connector $\texttt{Loop}_1$ and all its successors) in the corresponding process model S since `A` is on the left side of connector $\texttt{Loop}_1$. Note that when representing a loop structure within a process structure tree T, we introduce a silent activity τ as direct successor of the respective `Loop` connector (cf. Fig. 2.2b). This way we ensure that any connector in T always has at least two successors.[5] In

[5] Connectors of type `Seq, AND` and `XOR` represent a relation between its successors and require at least two successors. In case of "empty" branches in an XOR branching, again silent nodes

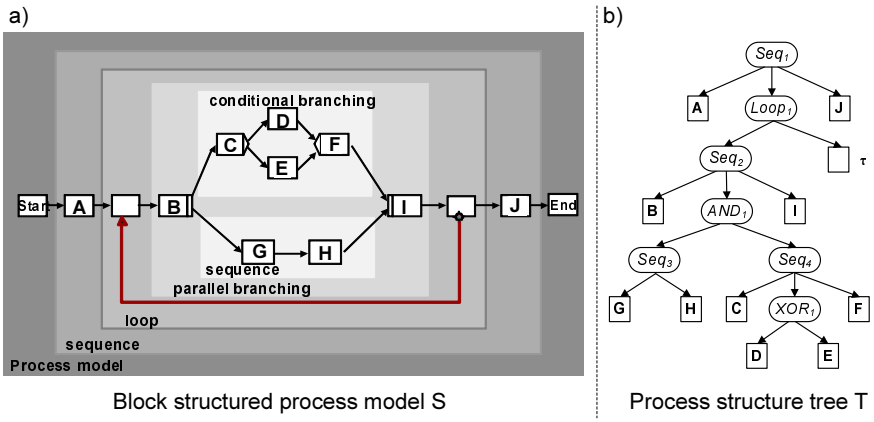


Figure 2.2: Process Model S and its corresponding process structure tree T

a process structure tree, nodes correspond to leaves, while connectors are non-leaves. Further, a process structure tree has a unique root node without incoming edges.[6]

When comparing a block-structured process model and its corresponding process tree, the main advantage of the latter is that a process structure tree contains fewer unnecessary silent activities, i.e., it provides a clear picture of the process model's structure and the relations between the activities. In the following chapters of the thesis, *activities* refer to the *nodes of a process structure tree*; i.e., an activity is either a labeled node or an un-labeled node, which exist to represent the loop structure in its corresponding process structure.

Definition 3 (Ancestor and Subtree) *Let $T = (N, C, CT, E, l)$ be a process structure tree. Let $a, b \in N \bigcup C$ be two elements of T. Then:*

1. *a is an **ancestor** of b $(a \prec b)$: $\Leftrightarrow$ There exists a path from a to b.*
2. *$\mathcal{A}_T(a) = \{b | (b \in N \bigcup C) \wedge (a \prec b)\}$ is denoted as descendant set of element $a \in N \bigcup C$.*
3. *A **subtree** T' of tree T is a process structure tree $T' = (N', C', CT', E', l')$ with the following properties:*
   - *$(N' \subseteq N) \wedge (C' \subseteq C)$*
   - *$\exists a \in N' \bigcup C' : N' \bigcup C' = \mathcal{A}_T(a) \bigcup \{a\}$; i.e., a is the root element of T'*
   - *$E' = \{(a, b) | a, b \in N' \bigcup C' \wedge (a, b) \in E\}$*

may be introduced for representing them.

[6] A block-structured process model can also be expressed by a shuffle regular expression [47]. For example, the process model S described in Fig. 2.2a can be expressed as a shuffle regular expression $A.(B.((G.H)||(C.(D+E).F)).I)^*.J$ as well. In this expression "$+$" is choice, "$.$" represents a concatenation, "$*$" denotes a Kleene star, and "$||$" is the shuffle operator in automata theory [77].

A sub-tree $T' = (N', C', CT', E', l')$ of process structure tree $T = (N, C, CT, E, l)$ is a connected fragment of T which contains a unique root element $a \in N \bigcup C$ and all its descendants. As example consider Fig. 2.2b: Activities `D` and `E`, their connector $\texttt{XOR}_1$, and the edges linking them can form a subtree of T. Since the process model S represented by T is block-structured, any subtree of T corresponds to a **block** of S; i.e., a process structure tree and its hierarchical decomposition into subtrees correspond to a block-structured process model and its hierarchical decomposition into blocks [204]. Given an element $e \in N \bigcup C$ in a process structure tree and taking Def. 3, we are able to construct a subtree $T(e)$ by identifying its descendant set $\mathcal{A}_T(e)$ and the edges linking them. In our example, the subtree of connector $\texttt{Seq}_4$ is a tree containing nodes `C,D,E` and `F`, connectors $\texttt{Seq}_4$ and $\texttt{XOR}_1$, and the edges connecting these elements.

2.1.3 Process Instance

A *process instance* "represents a concrete case in the operational business of a company ... Each process model acts as a blueprint of a set of process instances" [225]. For one particular process model, multiple instances may be created. Each of them represents a particular business case. Logically, such an instance corresponds to a process model being annotated with state information. Regarding block-structured process models, for example, a process model instance (*process instance* for short) is defined "by the current marking of its nodes and edges as well as its execution history" [141]. Fig. 2.3 shows a process model instance which is based on the process model from Fig. 2.1. Here, activities `A,B,C` and `G` are completed. Activity `D` is still running, while activity `E` was skipped since it is contained in a non-selected branch. Finally, activity `H` is activated, i.e., this activity is ready for execution.

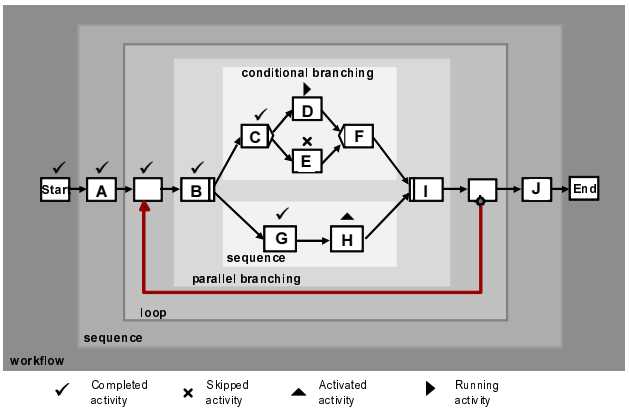


Figure 2.3: Process model instance

In this thesis we do not formally describe the operational semantics for block-

structured process models, but refer to [141, 152, 140] instead. As the process patterns used in block-structured process model can be easily mapped to WS-BPEL [21] or Petri Nets [193, 196], we could also describe the operational semantics of block-structured process models based on these languages. For example, Fig. 2.4 shows the corresponding[7] Petri Net of the block-structured process model from Fig. 2.1. Generally, the enactment of activities and the execution semantics of a process model can be described by the firing rules of Petri Net [126]. In principle, we can consider block-structured process models as a subclass of Workflow Nets [196], for which the net models follow the discussed structuring constraints.

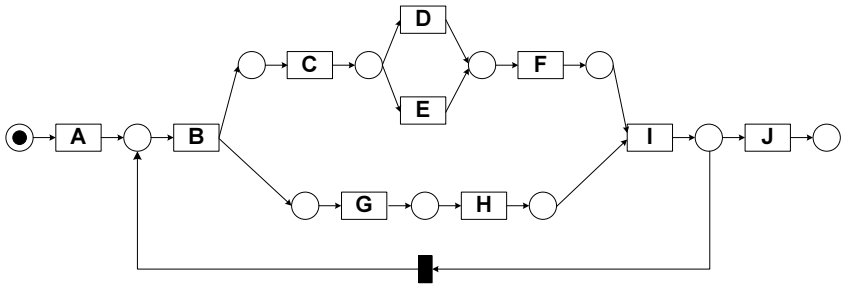


Figure 2.4: Petri Net of process model in Fig. 2.1

Similar to a Workflow Net, we consider a block-structured process model S as being **sound** if and only if the following properties hold:

1. *Proper completion* (i.e., when the EndFlow node becomes enabled, all other nodes cannot be activated anymore)

2. *Absence of deadlocks* (i.e., as long as the EndFlow node has not been enabled, there is no execution situation in which no node is activated)

3. *Absence of dead tasks* (i.e., there exists no node, which can be never activated)

For a formal description of these properties, we refer to [196, 187, 140]. Note that the soundness of a block-structured process model has the same requirements as the soundness of its corresponding Petri Net [126, 187]. In the following, we consider soundness as fundamental requirement any process model should satisfy as prerequisite for its proper execution and further analyses [187, 141]. How to verify the soundness of a process model, however, is out of the scope of this thesis (cf. [140, 141, 152, 187, 196] for respective techniques, which often employ reachability analyses as known from Petri Nets). In the following, let $\mathcal{P}$ denote the set of all block-structured as well as sound process models.

[7]Precisely speaking, the ADEPT model (cf. Fig. 2.1) and the Petri Net model (cf. Fig. 2.4) are not exactly the same; i.e., the execution semantics of the conditional branching is different. We omit the detail discussions here and refer readers to [185, 148].

2.1.4 Process Execution Log

In practice, hundreds up to thousands of process instances may be created based on one particular process model. In PAISs (e.g., ERP systems and workflow management systems) information about the past execution of process instances is usually maintained in *execution logs*. For each executed activity, such execution log contains information like the point in time the activity was stated or completed, the process instance it belongs to, the user who performed the activity, and so on [195]. Table 2.1.4 gives a simple example of an execution log based on the process model from Fig. 2.1. Genearlly, an execution log contains important run-time information about a process model and is used for various kinds of analyses [195, 197]. We provide a general overview of techniques for mining process execution logs in Section 2.5.

Activity	Related Instance	Event	User	Time
A	I_1	Activated	Chen Li	12:10:13
A	I_1	Running	Chen Li	12:10:57
A	I_2	Activated	Chen Li	12:11:11
A	I_2	Running	Chen Li	12:11:49
A	I_1	Completed	Chen Li	12:12:34
B	I_1	Activated	Edward Fang	12:12:35
B	I_1	Running	Edward Fang	12:12:50
A	I_2	Completed	Chen Li	12:14:02
B	I_2	Activated	Edward Fang	12:14:23
A	I_3	Activated	Jason Zhang	12:14:46
A	I_3	Running	Jason Zhang	12:15:26
B	I_1	Completed	Edward Fang	12:16:23
C	I_1	Activated	Chen Li	12:16:59
C	I_1	Running	Chen Li	12:17:48
B	I_2	Running	Edward Fang	12:18:21
A	I_3	Completed	Jason Zhang	12:19:18
		...		

Table 2.1: Examples of an execution log

From an execution log, we also can extract *trace* information for each process instance. Based on their time stamp we can order the activities relating to one particular process instance into a sequence. For example, given the execution log from Table 2.1.4, we can obtain the (incomplete) activity sequences ABC for process instance I_1, AB for process instance I_2, and A for process instance I_3. Generally, we consider the notion of *trace* as *valid* and *complete* execution sequence of activities regarding a particular process instance. Formally, we define the notion of trace as follows:

Definition 4 (Trace) *Let $S = (N, E, NT, ET, l) \in \mathcal{P}$ be a sound and block-structured process model. Let further $t \equiv < a_1, a_2, \ldots, a_k >$ (with $a_i \in N$) be a sequence of activities. We denote t as a trace of S iff:*

- *t is valid, i.e., the given execution sequence is producible on S.*
- *t is complete, i.e., a_1 is executed immediately after the completion of the StartFlow node, and a_k is executed immediately before executing the EndFlow node.*

We define $\mathcal{T}_S$ as the set of all traces that can be produced by process model S.

We only consider traces that log events related to labeled activities, whereas events concerning silent activities are excluded (cf. Def. 1). As example consider process model S from Fig. 2.1. Activity sequences like `ABCDFGHIJ, ABGHCEFIJ`, and `ABGCDFHIBCEFGHIJ` constitute both valid and complete traces of S. Like most process mining algorithms, we assume that the behavior of process model S can be expressed in terms of its trace set $\mathcal{T}_S$. Note that $\mathcal{T}_S$ constitutes an infinite set if S contains loops.

2.2 Adaptive Process Management

The ability to effectively deal with process changes and process configurations has been identified as one of the most fundamental success factors in PAISs [124, 133, 149, 211, 213]. Considerable efforts have been made to make process models more flexible [217, 66, 141, 165, 50]. For example, configurable reference models allow to switch on/off process activities at runtime; i.e., they offer flexibility to skip some activities [165]. Late binding [1] and late modeling [109] techniques defer modeling decisions to runtime by enabling users to select fragments at runtime and to specify control dependencies between them on-the-fly. Declarative approaches [189, 133] further enhance flexibility by only providing a set of rules and constraints, so that users can compose a process model flexibly, while taking into consideration the defined constraints. One common property of all these approaches is that they presume a solid understanding of the process model. No matter how loosely a process model is defined in order to enhance flexibility, constraints or potential configurations need to be provided up-front. Consequently, they cannot cover all exceptions occurring in practice [211]. In the following, we discuss another category of process flexibility techniques, which enable dynamic modifications of a process model's structure during runtime.

2.2.1 Dynamic Process Changes

During the last decade, a variety of dynamic process adaptation techniques was introduced. These enable both process configurations at build time and process changes during runtime, while preserving system robustness and consistency [141, 187]. In the following, we first introduce the notion of process change:

Process change: A process change is accomplished by applying a sequence of high-level change operations to a given process model S [141]. Such operations structurally modify the initial process model by altering its set of activities and their order relations. Thus, each application of a change operation results in a new process model:

Definition 5 (Process Change and Process Variant) *Let $\mathcal{P}$ denote the set of possible process models and $\mathcal{C}$ be the set of possible process changes. Let $S, S' \in \mathcal{P}$ be two process models, let $\Delta \in \mathcal{C}$ be a process change, and let $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle$ be a sequence of changes performed on initial model S. Then:*

- *$S[\Delta\rangle S'$ iff Δ is applicable to S and S' is the (sound) process model resulting from the application of Δ to S.*
- *$S[\sigma\rangle S'$ iff $\exists\ S_1, S_2, \ldots S_{n+1} \in \mathcal{P}$ with $S = S_1$, $S' = S_{n+1}$, and $S_i[\Delta_i\rangle S_{i+1}$ for $i \in \{1, \ldots n\}$. We also denote S' as **process variant** of S.*

Examples of high-level change operations include *insert activity*, *delete activity*, and *move activity* as implemented in the ADEPT change framework [141]. While *insert* and *delete* modify the set of activities in a process model, *move* changes activity positions and thus the structure of the process model. A formal semantics of these and other change patterns can be found in [160]. For example, $move(S, \texttt{A,B,C})$ moves activity `A` from its current position within process model S to the position after activity `B` and before activity `C`. Operation $delete(S, \texttt{A})$, in turn, deletes activity `A` from process model S. Finally, $insert(S, \texttt{A,B,C})$ adds activity `A` to the position after activity `B` and before activity `C`.[8] If some additional constraints (e.g., concerning the state of a running process) are met, the high-level change operations depicted in Table 2.2 will be also applicable at process instance level. Issues concerning the correct use of these change operations, their generalization, and formal pre-/post-conditions are outside the scope of this thesis and are described in [141].

Though the depicted change operations are discussed in relation to the ADEPT change framework (cf. Section 2.3), they are generic in the sense that they can be also applied in connection with other process meta models [160, 211, 215]. For example, a process change as realized in the ADEPT framework can be mapped to the concept of life-cycle inheritance known from Petri Nets [187]. We refer to ADEPT since it covers by far most high-level change patterns and change support features when compared to other adaptive PAISs [211]. Furthermore, with the AristaFlow BPM Suite [37], an industrial-strength version of the ADEPT technology has emerged, which has been applied in a variety of application domains[94].[9]

In addition to an *execution log* (cf. Section 2.1.4), adaptive PAISs maintain ***change logs*** [153, 157]. A change log documents the sequence of changes applied to a process model during its configuration and adaptation respectively.

Definition 6 (Change Log) *Let $\mathcal{P}$ denote the set of possible process models. Let $S \in \mathcal{P}$ be a process model and let $S_i \in \mathcal{P}$ for $i \in \{1, \ldots, n\}$ be its variants. A change log cL is defined as a set $\{\sigma_i \big| S[\sigma_i\rangle S_i\}$ (σ_1 denotes a sequence of change operations transforming S into S_i).*

[8] The ADEPT change framework supports change patterns like inserting, deleting, and moving activities and process fragments. It further supports additional change patterns as described in [211].

[9] Visit www.aristaflow-forum.de for details.

Change Operation Δ on S	opType	subject	paramList
insert(S, X, $\mathcal{A}, \mathcal{B}$, [sc])	insert	X	S, $\mathcal{A}, \mathcal{B}$, [sc]
Effects on S: inserts activity X between activity sets $\mathcal{A}$ and $\mathcal{B}$. It is a conditional insert if [sc] is specified (i.e. [sc] = XOR)			
delete(S, X, [sc])	delete	X	S, [sc]
Effects on S: deletes activity X from S, i.e. X turns into a silent activity. [sc] is specified ([sc] =XOR) when blocking the branch with X, i.e. the branch which contains X will not be activated.			
move(S, X, $\mathcal{A}, \mathcal{B}$, [sc])	move	X	S, $\mathcal{A}, \mathcal{B}$, [sc]
Effects on S: moves activity X from its original position in S to another position between activity sets $\mathcal{A}$ and $\mathcal{B}$. (It is a conditional insert if [sc] is specified)			

Table 2.2: *Examples of High-Level Change Operations*

Fig. 2.5b shows an example of a change log. This log is composed of nine change log instances $cl_{I_1} - cl_{I_9}$. The first change log instance cl_{I_1}, for example, consists of a move operation ($op1$) and an insert operation ($op2$).

2.2.2 Change Patterns

Like workflow patterns [193], which describe common modeling constructs of a process meta model (e.g., sequence, AND/XOR branching [193]), ***change patterns*** have been introduced to describe common change operations and change features respectively [211]. In total, 14 process adaptation patterns were identified with well-defined pre/post-conditions and well-defined semantics [215, 211, 160]. These change patterns include basic process change operations like deletion, insertion and movement of activities (and process fragments respectively) (see Table 2.2), as well as advanced patterns like *swap process fragments* or *embed process fragment in loop* [211]. In [125], additional patterns were identified by separating changes at buildtime and runtime, or differentiating between temporary changes and permanent ones. In the context of this thesis, we consider three basic change patterns: activity insertion, activity deletion and activity movement (cf. Table 2.2).

1. Based on these three basic change patterns, we can construct most of the high-level changes. For example, *swap process fragments* can be realized based on move operations. *Replace process fragment* can be realized by activity deletions plus activity insertions. Finally, as we will discuss in Chapter 4, the *embed process fragment in loop* pattern can be realized based on activity insertions in our approach.

2. Basic change patterns are more commonly supported by available tools than complex ones. Except the ADEPT change framework, which supports most

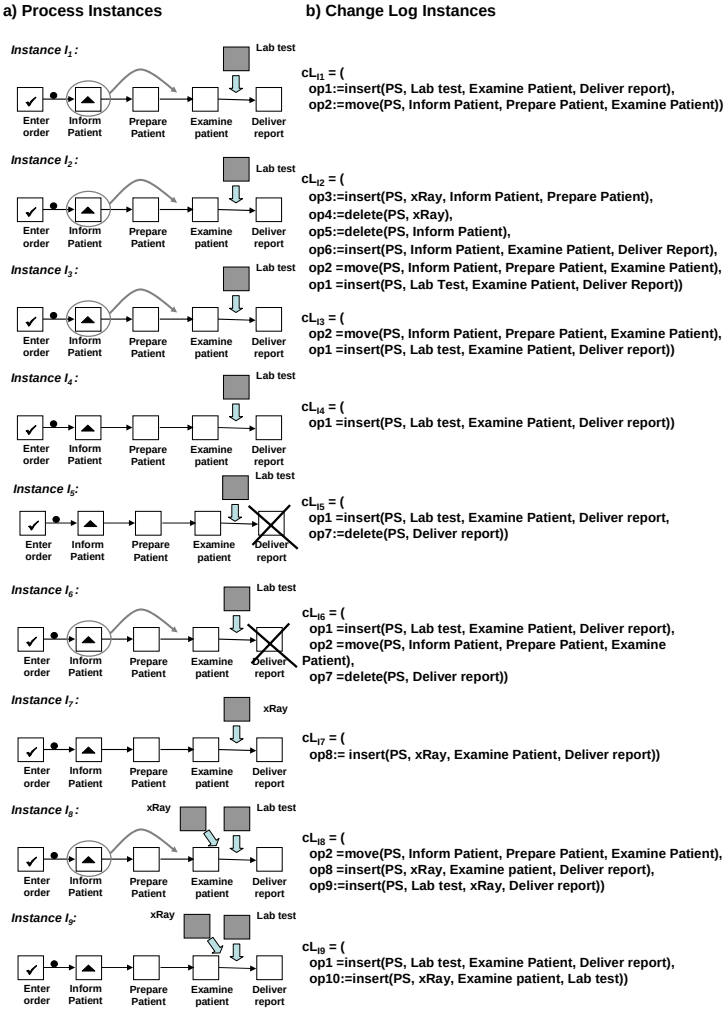


Figure 2.5: Modified Process Instances and Change Log Instances (taken from [62])

change patterns, available tools either support no high-level change patterns at all or only basic ones [211].

For these two reasons, we will focus on the three basic change patterns *insert, delete*, and *move* activities.

2.3 The ADEPT Process Management Technology

The work provided by this thesis has been accomplished in the context of the ADEPT project. This section gives some insights into ADEPT, which also help to position the contribution of this thesis.

The ADEPT process management project dates back to the late 90's. Its target was to develop a next generation process management technology, which is more powerful and flexible than contemporary process management systems are [37, 148]. After a decade of research and development, the ADEPT2 process management system emerged, which enables flexible execution of process instances. ADEPT2 is considered as one of the leading adaptive process management system nowadays. The ADEPT2 framework applies a rigor "correctness-by-construction" principle, and enables ad-hoc changes of single process instances during runtime without losing control [141, 37]. It also supports changes at the process type level and their propagation to running instances if desired and possible [147]. In the meanwhile, an industry-strength version of the ADEPT2 process management technology, called AristaFlow BPM Suite, has become available for academic and industrial use [37]. The advanced flexibility support features have been demonstrated in numerous projects on advanced applications[94]. [10]

The ADEPT2 technology provides advanced features and properties within one system, which seem to exclude each other, but which are required for the support of a broad spectrum of processes: ease-of-use for end users and system developers, high flexibility through the support of non-trivial ad-hoc deviations at the process instance level, quick implementation of process changes through process schema evolution, and correctness guarantees enabling the robust execution of implemented processes [37, 148, 141, 142]. Since ADEPT supports ad-hoc deviations at both runtime and build time, and also guarantees the soundness (correctness) of the (resulting) process model in this context, it provides a powerful technology basis for mining process variants in the context of our research.

2.4 Process Lifecycle

Figure 2.6 shows the general lifecycle of a flexible PAIS [213, 217, 218]. It starts with the design of a sound process model, based on which process instances can be created and executed to support respective business cases. The execution of these instances is documented in an execution log (cf. Definition 4). If needed, authorized users may deviate from the process model (e.g., by adding, deleting or moving process activities) at the level of individual process instances [141, 187, 216]. Respective instance changes, which capture the deviation from the original process model, are logged in a change log (cf. Definition 6). Based on the information from both execution and change logs, we can learn from past process executions and discover opportunities to optimize and evolve process

[10] Visit www.aristaflow-forum.de for details.

models [195, 197, 101, 98]. If, for example, a certain change happens over and over again at the process instance level, the process designer will be notified and assisted in adapting the original process model accordingly [141]. In case of long-running process instances, respective process type changes may be propagated to the instance level as well [147, 27].

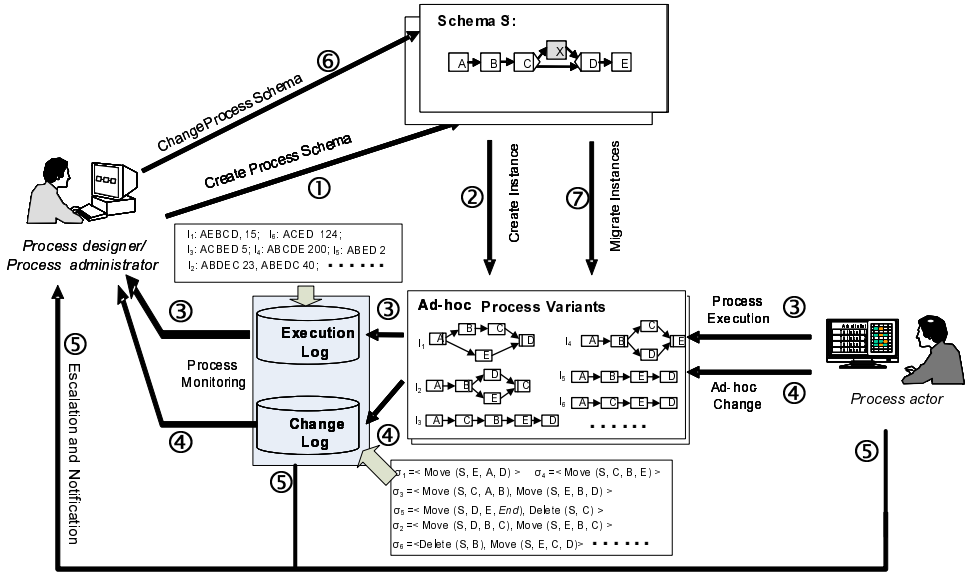


Figure 2.6: The Lifecycle of a PAIS According to [213]

Many efforts have been undertaken to enable PAISs to provide full lifecycle support and approaches like ADEPT2 [141, 148], CAKE2 [119], WASA2 [224, 207, 223], TRAMs [87], Worklets/Exlets [1, 2], and YAWL [192] have emerged in this context (for a comprehensive overview see [154, 211]). They all target at providing users the flexibility to dynamically change the processes running in the PAISs according to real-world situations.

As discussed in Chapter 1, we focus on the *discovery* of process models by learning from past process changes and executions; i.e., we focus on Step 5 of the PAIS lifecycle (cf. Fig. 2.6). Particularly, we are interested in learning from change logs or directly from process variants (if change logs do not exist). When compared to traditional mining techniques, which focus on the analyses of execution logs, analyzing change logs (or process variants) provides additional opportunities (see [61, 62] for details). Chapter 8 will further compare mining techniques based on execution logs and on change logs.

Before we deal with the analysis of process change logs (or process variants resulting from their application to the original process model), Section 2.5 introduces traditional *process mining* techniques, which particularly focus on the analysis of execution logs [195, 197, 39].

2.5 Process Mining

Process mining describes a family of analysis techniques exploiting the information recorded in these logs [195, 197, 222, 39, 221, 200]. Typically, respective approaches assume that it is possible to sequentially record events such that each event refers to an activity and is related to a particular process instance (i.e., trace in our context, cf. Def. 4).

2.5.1 Overview

Process mining addresses the problem that most "process owners" have very limited information about what is actually happening in their organization. In practice there is often a significant gap between what is prescribed or supposed to happen, and what actually happens. Only a concise assessment of the organizational reality, which process mining strives to deliver, can help in verifying process models [126], and ultimately be used in a process redesign effort.

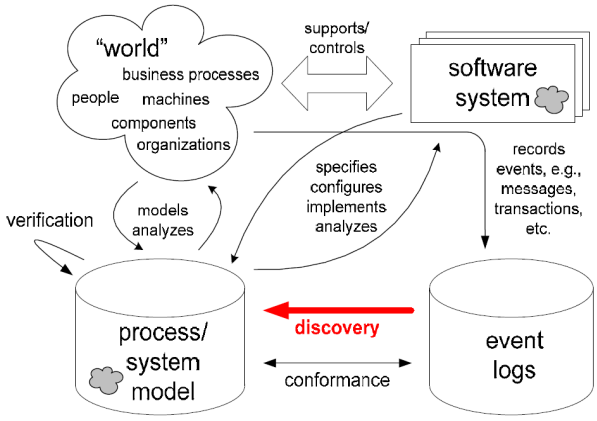


Figure 2.7: Overview of process mining [8]

There exist three major classes of process mining techniques as indicated in Fig. 2.7. Traditionally, process mining has focused on *process discovery*, i.e. on deriving information about the original process model, the organizational context, and the execution properties from execution logs (i.e., event log in Fig. 2.7) [195]. An example of a technique addressing the control flow perspective is the alpha algorithm [197]. It can construct a Petri net model [185] describing the behavior observed in an execution log. The Multi-Phase Mining approach [200] can be used to construct an Event-driven Process Chain (EPC) [171] based on similar information. First work regarding the mining of other model perspectives (e.g.,

[8] This figure is cited from the tutorial talk "Process Mining Tutorial: Beyond Business Intelligence" given by Prof. W.M.P van der Aalst in BPM'08 conference.

organizational aspects [190, 112]) and data-driven process support systems (e.g., case handling systems) [60] has been introduced as well.

Another line of process mining research is *conformance testing* [168, 5]. Its aim is to analyze and measure discrepancies between the model of a process and its actual execution (as recorded in event logs). This can be used to indicate problems. Finally, *log-based verification* does not analyze execution logs with respect to the original model, but rather checks the log for conformance with certain desired or undesired properties. For example, one can check whether or not an process instance is compliant with certain laws or corporate guidelines [5].

At this point, there exist mature tools such as the ProM framework, which can be applied to real process execution logs, while covering the whole spectrum depicted in Fig. 2.7 [201].

2.5.2 Illustrating Example

We briefly introduce one selected process mining algorithm, namely the Alpha Algorithm [197], to illustrate how process mining works in general. The Alpha Algorithm can be considered as one of the first algorithms suggested for process mining, and is often used for illustration purposes (though there exist more mature algorithms in the meanwhile).

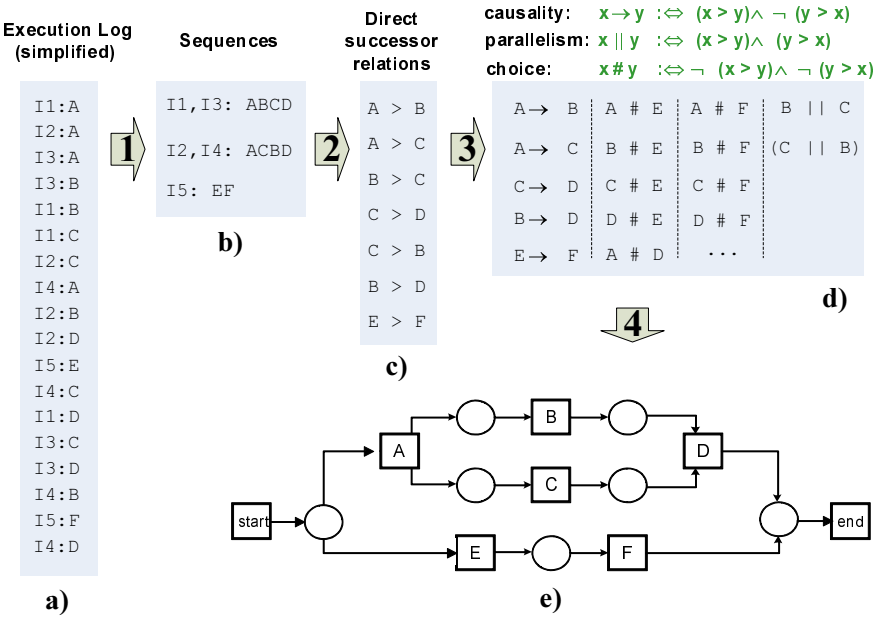


Figure 2.8: Illustrative example for process mining algorithms

Fig. 2.8a first shows a simplified version of an execution log. This log abstracts from time, event type, and actors, but limits the information to the order in which

activities were executed. The log further contains information about five process instances. In each of these process instances, some of the six activities (`A,B,C,D,E` and `F`) were executed in different orders.

The execution log is first transformed into activity sequences according to the order in which the activities were executed in each process instance (i.e., traces (cf. Def. 4)). In our example, process instances I_1 and I_3 have activity sequence `ABCD`, I_2 and I_4 have sequence `ACBD` and instance I_5 has activity sequence `EF`. Based on these activity sequences, we can identify the direct successor relationship between activities. For example, in activity sequence `ABCD` we can see that `B` is a direct successor of `A` (denoted as $\texttt{A} > \texttt{B}$), `B` is a direct successor of `C`, ($\texttt{B} > \texttt{C}$) and `C` is a direct successor of `D` ($\texttt{C} > \texttt{D}$). The direct successor relationships obtained by the execution log are depicted in Fig. 2.8c.

Based on the direct successor relationships, we can define three types of ordering relations: causality ($x \rightarrow y$), parallelism ($x||y$), and choice ($x\#y$). For example, we obtain causality relationship $x \rightarrow y$ if y is a direct successor of x, but not vice versa (i.e., $(x > y) \wedge (y > x)$). The rules for deriving the three types of ordering relations as well as the results of our example are shown in Fig. 2.8d). Based on these ordering relations, we can obtain a process model described in terms of a Petri Nets (see Fig. 2.8e). Clearly, the alpha algorithm is more advanced than what we described in this case and is able to handle more complex scenarios. Due to space limitations, we omit these complexity and technical details, and refer to [197] instead.

This simple example shows how process mining works in general. Though advanced algorithms like heuristic mining [221] or genetic mining [39] can deal with more complex scenarios (including noise in logs or duplicated tasks), they basically share similar design principles.

In principle, process mining algorithms are also applicable in the context of this thesis, i.e., for mining process variants. We will briefly discuss this in Chapter 3, and systematically compare traditional process mining algorithms with our algorithms in Chapter 8.

3

Mining Process Variants: Challenges and Goals

3.1 Introduction

This chapter tries to answer our first research question (cf. Section 1.2):

What are fundamental challenges in mining process model variants? Are existing mining techniques suitable for realizing the goal of reducing process configuration efforts?

To be more precise, Section 3.2 first describes the scientific challenges that emerge in respect to the mining of process model variants. Section 3.3 then discusses the specific goals of our research and explains why traditional process mining techniques cannot satisfy them.

3.2 Challenges for Mining Process Variants

Before we deal with issues related to the the mining of process variants, we discuss basic aspects concerning the representation of process changes. First, we explain why process variant mining cannot be solely based on execution logs, but necessitates the consideration of applied changes as well. Second, we sketch why it is beneficial to express changes in terms of high-level change operations (cf. Def. 2.2) rather than low-level change primitives (e.g., to add/delete nodes and edges). Third, we discuss how the application of high-level change operation affects execution behavior of a process model.

3.2.1 Complementary Nature of Change and Execution Logs

Adaptive PAISs support ad-hoc deviations at the process instance level and record them in *change logs* (cf. Section 2.2.1). Thus, they provide additional information when compared to traditional PAISs which only record *execution logs* (cf. Section 2.1.4). Change logs and execution logs document different run-time information on adaptive process instances and are not interchangeable. Even if the original

process model is given, it will be not possible to convert the change log of a process instance to its execution log or vice verse.

As example, take the simplified medical treatment procedure as depicted in Fig. 3.1a: a patient is admitted (activity `admit`) to a hospital, where he first registers (activity `register`), then receives treatment (activity `receive treatment`), and finally pays (activity `pay`). Assume that, due to an emergency situation, for one particular patient, we first want to start the treatment of this patient and allow him to register later during treatment. To represent this exceptional situation in the process model of the respective instance, the needed change would be to move activity `receive treatment` from its current position to a position parallel to activity `register`. This change leads to a new model S', i.e., $S[\sigma\rangle S'$ with $\sigma =< move(S,$ `reveive treatment`, `admit`, `pay`$) >$. In addition, the execution log e for this particular instance can be $e =<$ `admit, receive treatment, register, pay` $>$ (cf. Fig. 3.1b). If we solely consider process model S and its execution log, it will be not possible to determine this change because the process model which can produce such execution log is not unique. For example, a process model with the four activities organized in four parallel branches could produce this execution log as well. On the contrary, it is generally not possible to derive the execution log from a change log, since execution behavior of S' is also not unique. For example, a trace $<$ `admit`, `register`, `receive treatment`, `pay` $>$ is also producible on S'. Consequently, change logs provide additional information when compared to pure execution logs.

3.2.2 Why Do We Need High-level Change Operations?

We now discuss why we prefer high-level change operations for expressing process model changes rather than low-level change primitives (i.e., low-level changes of process graphs at edge and node level). Consider Fig. 3.2. Its left hand side shows an original process model S which consists of a parallel branching, a conditional branching, and a silent activity τ (depicted as empty node) connecting these two blocks. Assume that two different high-level change operations are applied to S resulting in models S_1 and S_2 respectively: Δ_1 moves activity `C` from its current location to the position between activities `A` and `B`, which leads to process model

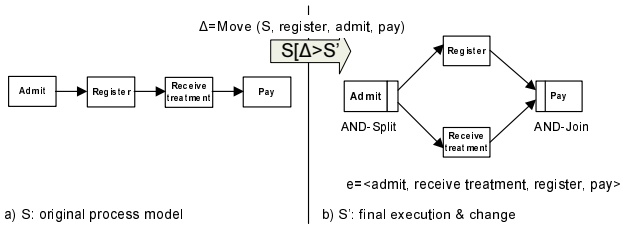


Figure 3.1: Change Log and Execution Log are not interchangeable

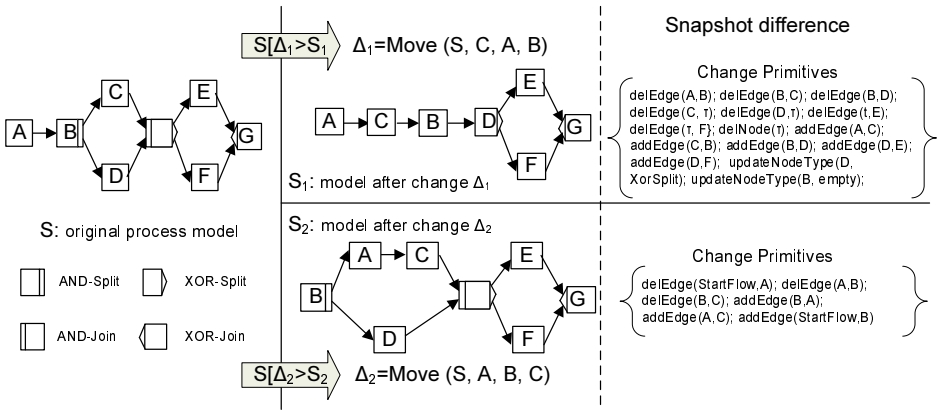


Figure 3.2: High-level change operation and corresponding change primitives

S_1, i.e., $S[\Delta_1\rangle S_1$ with $\Delta_1 = move(S, \texttt{C}, \texttt{A}, \texttt{B})$. Δ_2, in turn, moves activity `A` to the position between activities `B` and `C`, i.e. $S[\Delta_2\rangle S_2$ with $\Delta_2 = move(S, \texttt{A}, \texttt{B}, \texttt{C})$. Fig. 3.2 additionally depicts the change primitives representing snapshot differences between S and models S_1 and S_2 respectively.

Overall, using high-level change operations offers the following advantages:

1. High-level change operations contribute to guarantee model soundness: i.e., the application of a high-level change operation to a sound model S will result in another sound model S' if certain pre-conditions are met [141]. This also applies to our example from Fig. 3.2. By contrast, when applying one single change primitive (e.g., deleting an edge in S) soundness cannot be guaranteed in general. For example, if we delete any of the control edges in S, the resulting process model will be not structurally sound (cf. Def. 5).

2. High-level change operations provide richer syntactical meanings than change primitives. Generally, a high-level change operation is built upon a set of change primitives which collectively represent a complex modification of a process model. As example take Δ_1 from Fig. 3.2. This high-level change operation requires *15* change primitives for its realization (deleting edges, adding edges, deleting the silent activity, and updating the node types).

3. An important aspect, not discussed so far, concerns the number of change operations needed to transform model S into target model S'. For example, we need to apply only *one* move operation to transform S to either S_1 or S_2 (cf. Fig. 3.2). However, when using change primitives, migrating S to S_1 necessitates *15* change primitives, while the second change Δ_2 can be realized based on *6* change primitives. This example shows that change primitives do not provide an adequate means to determine the difference

between two process models. Thus the required number of change primitives cannot adequately represent the efforts for process model transformations.

3.2.3 The Challenge to Derive High-level Changes

After sketching the benefits coming with high-level change operations, this section discusses the challenges to be tackled when deriving them. When comparing two process models, the change primitives needed for transforming one model into another can be easily determined by performing two snapshots and a delta analysis on them [93]. An algorithm to minimize the number of change primitives is given in [153]. However, when trying to derive the high-level change operations needed for model transformation, a number of challenges occurs. As example consider Fig. 3.3:

1. When performing two delete operations on S (i.e., $\Delta_1 = delete(S, \mathtt{B})$ and $\Delta_2 = delete(S, \mathtt{C})$), we obtain new model S''; i.e., $S[\sigma\rangle S''$ with $\sigma =< \Delta_1, \Delta_2 >$, as well as an "undetectable" intermediate model S' with $S[\Delta_1\rangle S'$ and $S'[\Delta_2\rangle S''$. When examining the change primitives corresponding to each high-level change operation, we first need to add edge (`A,C`) after the first *delete* operation Δ_1, and remove this edge (`A,C`) when applying the second *delete* operation Δ_2. However, when performing a delta analysis for the original process model S and the resulting one S'', the two change primitives (addEdge(`A,C`) introduced by the first *delete* operation and delEdge (`A,C`) introduced by the second one) jointly have no effect on the resulting process model S'', i.e., they cannot be detected by snapshot analysis. Consequently, deriving high-level change operations based on change primitives would be challenging because the change primitives required for realizing every high-level change do not always appear in the snapshot differences between original and resulting model. In Fig. 3.3, none of the two change primitive sets associated with Δ_1 or Δ_2 constitute a sub-set of the change primitive set associated with overall change σ.

2. Even if there exists only one high-level change operation, it remains difficult to derive it by purely analyzing change primitives. For example, in Fig. 3.2 the delta algorithm shows that *15* change primitives are needed to transform S into S_1. However, the depicted changes can be also realized by just applying one high-level move operation to S.

For these reasons it is difficult to derive high-level change operations between two process models solely by analyzing change primitives. We will introduce a method based on boolean algebra to tackle this challenge in Chapter 5.

3.3 Goals for Mining Process Variants

This section discusses the major goal in respect to the mining of process variants, namely to derive a *reference process model* from a collection of process variants.

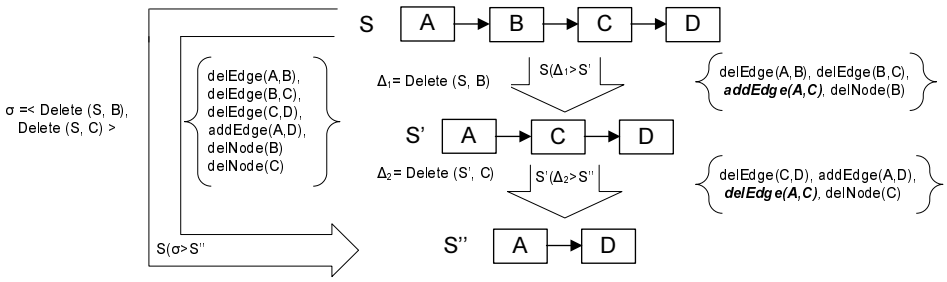


Figure 3.3: Non-detectable Change Primitives

This shall be done in a way such that the existing variants (as well as future ones) can be efficiently configured out of the discovered model. We measure efforts for corresponding process configurations in terms of the number of high-level change operations needed to transform the discovered reference model into the respective model variant. The challenge is to find a reference model such that the *average* number of high-level change operations needed (i.e., the *average distance*) to transform the discovered model into any variant becomes minimal with respect to the given variant collection.

To make this more clear, we first compare process variant mining with traditional process mining [195]. The latter (cf. Section 2.5) has been extensively studied in literature. Its key idea is to discover a process model by analyzing the execution behavior of (completed) process instances as captured in execution logs (cf. Section 2.1.4) [195]. Different mining techniques like alpha algorithm [197], heuristics mining [221] and genetic mining [39] have been proposed in this context. Obviously, input data for traditional process mining differs from the one of process variant mining. While traditional process mining operates on *execution logs*, process variant mining should be based on a collection of process model variants.

Of course, such high-level consideration is not sufficient to prove that existing mining techniques do not provide optimal results with respect to the aforementioned goal. In principle, existing process mining techniques [197, 39] can be applied to our problem as well. For example, we could derive all traces producible by a given collection of process variants [231] and then apply existing mining algorithms to them. To make the difference between process mining and process variant mining more evident, we consider *behavioral similarity* as well as *structural similarity* between two process models in the following.

The behavior of a process model S can be represented by the set of traces (i.e., $\mathcal{T}_S$) it can produce. Therefore, two process models can be compared based on the difference between their trace sets [197, 231]. By contrast, biases can be used to express the (structural) distance between two process models [100], i.e., the minimal number of high-level change operations needed to transform one model into the other. While the mining of process variants focuses on structural

similarity, traditional process mining addresses behavioral issues. Obviously, this leads to different choices with respect to the design of the mining algorithms and also suggests different mining results.

Fig. 3.4 depicts two very simple examples. First, consider Example 1 which shows two process variants S_1 and S_2. Assume that 55 process instances are running on S_1 and 45 instances on S_2. We want to derive a reference process model such that the efforts for configuring the 100 process instances out of the reference model become minimal. If we focus on behavior, like existing process mining algorithms do [197], the discovered process model will be S; all traces producible on S_1 and S_2, respectively, can be produced on S as well, i.e. $\mathcal{T}_{S_1} \subseteq \mathcal{T}_S$ and $\mathcal{T}_{S_2} \subseteq \mathcal{T}_S$. We use *trace coverage* to measure to what percentage the traces producible by the variants are also producible by the reference model. We obtain 100% as trace coverage for S, i.e., all traces producible by the variants can be produced by the reference model S as well. However, if we adopt S as reference model and relink process instances to it, all instances running on S_1 or S_2 will have a non-empty bias (bias can be measured in terms of the difference between reference model and variant models). More precisely, we would need to move `B` in S to either obtain S_1 or S_2; i.e., $S[\sigma_1\rangle S_1$ with $\sigma_1 = move(S, \texttt{B,A,C})$ and $S[\sigma_2\rangle S_2$ with $\sigma_2 = move(S, \texttt{B,C,D})$ (cf. Def. 5). Using the number of instances as weight for each variant, *average distance* between S and S_i $(i = 1, 2)$ is *one*; i.e., for each process instance we need on average one high-level change operation to configure S into S_1 and S_2, respectively.

By contrast, if we focus on biases we should choose S' as reference model. While no adaptations become necessary for the 55 instances running on S_1, we need to move `B` for the 45 instances based on S_2, i.e. $S'[\sigma'\rangle S_2$ with $\sigma' = move(S', \texttt{B,C,D})$. Therefore, average weighted distance between S' and variants S_i $(i = 1, 2)$ corresponds to *0.45*. Though S' does not cover all traces variants S_1 and S_2 can produce (i.e., $\mathcal{T}_{S_2} \not\subseteq \mathcal{T}_{S'}$, and we obtain trace coverage of S' as 55%), adapting S' rather than S as the new reference model requires by average less efforts for process configuration, since average distance between S' and the instances running on both S_1 and S_2 is 55% lower than when using S.

Regarding Example 2 from the bottom of Fig. 3.4, activity `X` is only present in variant model S_2, but not in S_1. When applying traditional process mining to this case, we obtain model S (with X being contained in a conditional branch). If we want to minimize average change distance, in turn, we need to choose S' as reference model. Note that we consider rather simple process models in Fig. 3.4 in order to illustrate basic ideas. In Chapter 8, we will provide a more systematic comparison based on a large amount of process variants with complex structure as well.

Our discussions on the difference between behavioral and structural similarity also indicate that current process mining algorithms do not consider structural similarity based on bias and change distance (we quantitatively compare our mining approach with existing algorithms in Chapter 8). First, a fundamental requirement for traditional process mining concerns the availability of a critical number of instance traces. An alternative method is to enumerate all traces the

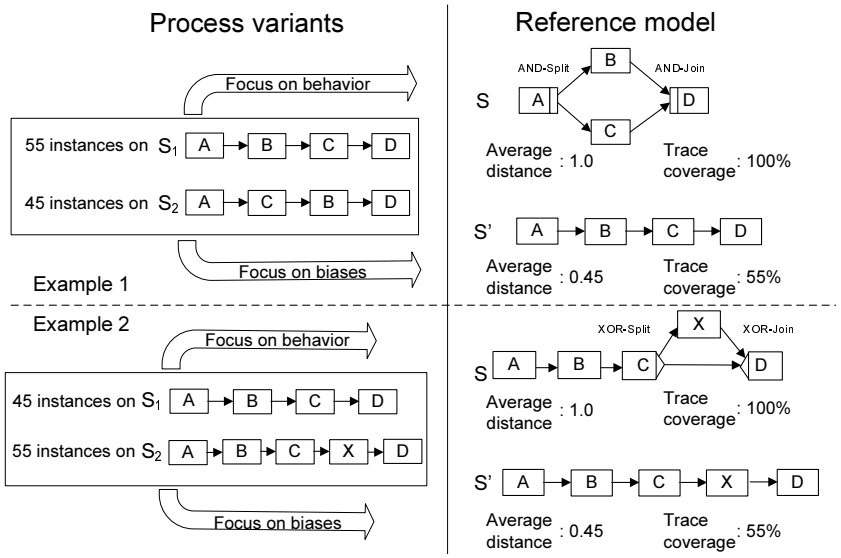


Figure 3.4: Mining focusing either on behavior or on their structure

process variants can produce (if it is finite) to represent the process model, and to use these traces as input source for traditional process mining algorithms. Unfortunately, this does also not satisfy our need to minimize average distances since it focuses on covering behavior as captured in execution logs (see Examples 1 and 2 from Fig. 3.4, and we will provide a detailed comparison in Chapter 8). Clearly, enumerating all the traces would be also a tedious and expensive task. For example, if a parallel branching block contains five branches and each branch contains five activities, the number of traces for such structure will be $(5 \times 5)!/(5!)^5 = 623,360,743,125,120$.[1]

Note that this Chapter explained our design goal and motivated our approach specially focusing on model adaptations in a rather informal and qualitative way. In Chapter 8, we will provide a detailed and quantitative evaluation of our approach based on formal evaluation criteria.

3.4 Summary

We have motivated the need for process variant mining, discussed its major goals as well as relevant technical issues, and elaborated its differences when compared to traditional process mining. We believe that process variant mining will enable learning from past process adaptations. We have re-motivated the goal of process variant mining: by mining a collection of process model variants, we want to

[1] Not all process mining algorithms require first enumerating the trace set of process variants, we discuss this issue in detail in Chapter 8.

discover a reference process model which can be easily configured into the process variants.

We have further briefly compared process variant mining with traditional process mining. This comparison has shown that traditional process mining does not fully satisfy the need for deriving a process model which is easy configurable. This justifies the efforts for designing specific algorithms for process variant mining. In Part II, we will introduce several algorithms for mining process variants, and in Chapter 8, we will give a systematic comparison between process mining and process variant mining.

Part II

Representing, Comparing & Mining Process Variants

4

Representing Block-structured Process Models as Order Matrices

4.1 Introduction

The ability to manage, analyze and diagnose business process models is getting increasingly important. Generally, it is not always sufficient to just consider the nodes and edges of a process model. Instead, more sophisticated and mathematical representations are needed to enable advanced analysis and processing options. In this context, matrix representations have been applied to various process analysis scenarios [181, 166, 195, 39]. For example, the marking of a Petri Net can be regarded as vector and its transition relations be described as matrix. Then the firing of a transition can be expressed in terms of matrix multiplication [187]. Based on this we can easily analyze properties like free-choice or liveness [39]. In process mining, *causal matrices* are used to represent relationships between transitions in Petri Nets. Causal matrices are also applied in the context of genetic process mining to discover a process model which captures the execution traces of a given process instance collection best [39]. In a more general way, we can consider process models as directed graphs and represent them by their *adjacency matrices* for various graph analyses (e.g., reachability analysis, derivation of minimal spanning tree, or graph pattern discovery [166, 181]). Finally, matrices are used to classify, cluster or associate graph data in various data mining fields [181].

So far, most matrix representations of process models (or graphs in general) have focussed on nodes and edges. However, in scenarios in which the ability to support process adaptation and configuration becomes increasingly important, these matrix representations cannot be directly applied [98]. For example, if we move one transition in a Petri net to another position all reachable states need to be re-computed, which results in a complete new matrix representation of this Petri net. Or, if we remove one activity from the adjacency matrix of a sound process model, we may obtain an erroneous matrix which does not represent a sound process model anymore.

In this chapter, we introduce the notion of *order matrix*. Basically, this matrix

represents all transitive relations between activities of a block-structured process model. In Chapter 5, this order matrix representation will act as a basis for measuring structural similarity between process models, and in Chapters 6 and 7 we will use it when mining process model variants. This chapter focuses on basic concepts, algorithms and formal properties of order matrices, while later chapters (Chapters 5, 6 and 7) apply these in several respects.

Chapter 4 is organized as follows. Section 4.2 first provides the formal definition of an order matrix and gives an illustrative example. Section 4.3 discusses benefits of using order matrices when compared to common tree representations of process models. We conclude with a summary in Section 4.4.

4.2 Basic Definition of an Order Matrix

In the context of process changes, one key feature is to maintain the structure of the unchanged parts of a process model [141, 187]. For example, when deleting an activity this should neither influence successors nor predecessors of this activity, and therefore also not their order relations. To incorporate this in our matrix-based model representation, rather than only looking at direct predecessor-successor relationships between activities (i.e. control edges), we consider the transitive control dependencies for each pair of activities. As example consider process model S in Fig. 4.1a and its corresponding process structure tree T in Fig. 4.1b. In the following, we will based the definition of the order matrix and its semantics on the process structure tree which we introduced in Section 2.1.2 (cf. Def. 2).

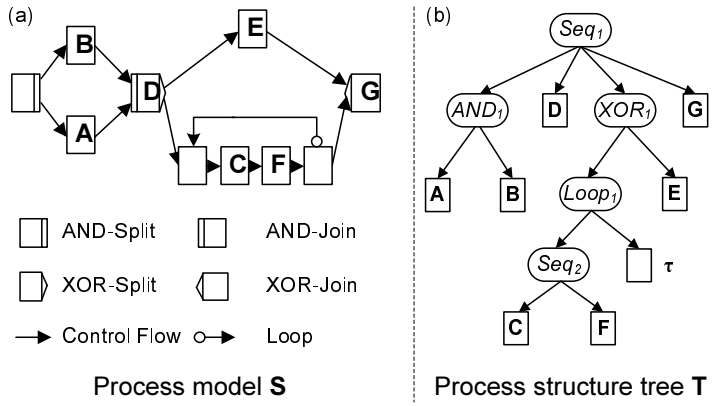


Figure 4.1: Process Model S and its corresponding process structure tree T

4.2.1 Nearest Common Ancestor

Before we formally introduce order matrix, we first need to introduce the concept of *nearest common ancestor* in a process structure tree:

Definition 7 (Nearest Common Ancestor) *Let $T = (N, C, CT, E, l)$ be a process structure tree and let $a, b \in N$ be two different nodes of T. Then, we denote connector $c \in C$ as nearest common ancestor $NCA(a, b)$ of these two nodes iff:*

- $c \prec a$ *and* $c \prec b$
- $\nexists c' \in C : c' \prec a$, $c' \prec b$ *and* $c \prec c'$

Note that the nearest common ancestor of two nodes always refers to a connector since nodes constitute leaves of the process structure tree and consequently cannot be ancestors. Here we have assumed that a process structure tree contains at least two nodes and one connector. Finding the nearest common ancestor in a tree is a well researched topic. Based on the algorithms presented in [70], we are able to compute the nearest common ancestor for any two nodes in a process tree T in linear time; i.e., $\mathcal{O}(n)$ with $n = |N \bigcup C|$.

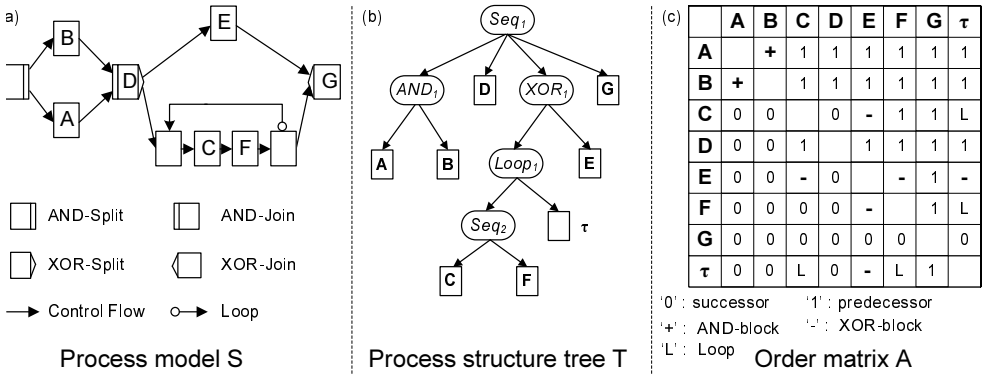
4.2.2 Representing a Process Model as Order Matrix

Based on the process structure tree T of a block-structured process model S and the concept of nearest common ancestor (cf. Def. 7), we can introduce the notion of *order matrix*, which uniquely represents a process structure tree and thus a block-structured process model (cf. Theorem 1).

Definition 8 (Order matrix) *Let S be a block-structured process model and let $T = (N, C, CT, E, l)$ be its process structure tree. A is called* ***order matrix*** *of T with $A_{a_i a_j}$ representing the order relation between activities a_i,$a_j \in N$, $i \neq j$ iff:*

- $A_{a_i a_j}$ *= '1' iff in T, a_i and a_j have as nearest common ancestor a "`Seq`" connector; a_i is contained in the left subtree of `Seq` and a_j in its right subtree.*
- $A_{a_i a_j}$ *= '0' iff in T, a_i and a_j have as nearest common ancestor a "`Seq`" connector; a_i is contained in the right subtree of `Seq` and a_j in its left subtree.*
- $A_{a_i a_j}$ *= '+' iff in T, a_i and a_j have as nearest common ancestor an "`AND`" connector.*
- $A_{a_i a_j}$ *= '-' iff in T, a_i and a_j have as nearest common ancestor an "`XOR`" connector.*
- $A_{a_i a_j}$ *= 'L' iff in T, a_i and a_j have as nearest common ancestor a "`Loop`" connector.*

Fig. 4.2 depicts the block-structured process model S, process structure T (from Fig. 4.1) and its corresponding order matrix A. Note that silent activity τ, which was introduced as the direct successor of connector $\texttt{loop}_1$ in T, is included



	A	B	C	D	E	F	G	τ
A		+	1	1	1	1	1	1
B	+		1	1	1	1	1	1
C	0	0		0	-	1	1	L
D	0	0	1		1	1	1	1
E	0	0	-	0		-	1	-
F	0	0	0	0	-		1	L
G	0	0	0	0	0	0		0
τ	0	0	L	0	-	L	1	

Figure 4.2: Block-structured process model S, Process structure tree T and order matrix A

in the order matrix A as well (cf. Section 2.1.2). This order matrix contains all five order relations from Def. 8. For example, activities `E` and `C` have as nearest common ancestor connector $\texttt{XOR}_1$. Thus, we assign '-' to matrix elements $A_{\texttt{EC}}$ and $A_{\texttt{CE}}$. Since activities `B` and `G` have as nearest common ancestor connector $\texttt{Seq}_1$, and `B` is on its left subtree while `G` is on its right one, we further obtain order relations $A_{\texttt{BG}}$ = '1' and $A_{\texttt{GB}}$ = '0' respectively.

Special attention should be paid to the order relations between silent activity τ and the other activities. Since τ is the direct successor of connector $\texttt{Loop}_1$, order relation 'L' indicates those nodes in T which are descendants of its corresponding `Loop` connector. Consequently the order relations between τ on the one hand and activities `C` and `F` on the other hand are set to 'L'. This implies that `C` and `F` are descendants of a `Loop` connecter and consequently implies that they are included in a loop block in the corresponding process structure and process model respectively. Note that the main diagonal of an order matrix is empty since we do not compute the nearest common ancestor of an activity with itself.

The following Theorem 1 states that an order matrix A can uniquely represent a corresponding process structure tree T.

Theorem 1 *Let $T = (N, C, CT, E, l)$ be a process structure tree. Let further $A_{|N|\times|N|}$ be the order matrix constructed based on T. Then: such order matrix A exists and is unique.*

Proof 1 *See Appendix A.2.*

Since a process structure tree T itself constitutes a unique representation of a block-structured process model S [204], and T can be uniquely represented by an order matrix A (cf. Theorem 1), order matrix A is a unique representation of S as well. Consequently, it is sufficient to analyze the order matrix of a block-structured process model. We make use of this in the following Chapters.

In [103] we provided two algorithms which transform a process model directly to its corresponding order matrix and vise versa without the need to compute the process structure tree first. This way we can reduce the complexity for such transformation to $\mathcal{O}(2m^2)$ (with $m = |N|$), which is lower than first transforming a process model into a tree ($\mathcal{O}(m)$), and then computing nearest common ancestor for every pair of activities ($\mathcal{O}(m^3)$).

4.3 Matrix and Tree: Representing Process Models from Different Perspectives

Various papers have discussed use cases for process structure trees [46, 204]. Generally a process tree provides a fine-grained decomposition of a process model that is suited for parsing, translation, and pattern discovery [204]. Process structure trees have been also used for speeding up process verification [205], for constructing process models based on process patterns [59], and for comparing process models [46]. As our order matrix (cf. Def. 8) is defined based on the process structure tree, we want to elaborate on the value and benefit added by the introduction of the order matrix.

4.3.1 Process Changes Made Easy

For any process model, its order matrix captures transitive order relations between activities. Based on this property, an order matrix can be used to specify process changes in terms of high-level change operations (e.g., delete, insert or move activities) [141, 187]. For example, if we *delete an activity* from an order matrix (i.e., remove its corresponding row and column), the resulting matrix will be a representation of a valid process model again since the order relations between all other activities remain unchanged. When considering Fig. 4.3, for instance, deleting activity `E` from order matrix A results in order matrix A', which again, represents a sound and block-structured process model S'. *Activity insertion* can be directly realized for order matrices as well. As example consider model S' in Fig. 4.3. Assume that we want to insert activity `E` as alternative choice (i.e., XOR) for the parallel block that comprises activities `A,B` and `D`. Clearly, we want to ensure that the resulting model remains sound and block-structured. This can be realized based on order matrix A' of S'. First, we add one row and one column to A' in order to capture the new activity `E`. Then we cluster `E` with block {`A,B,D`} based on order relation '-' (XOR block); i.e., we set $A_{\texttt{EA}}, A_{\texttt{EB}}$ and $A_{\texttt{ED}}$ to '-'. Further, we set order relations between `E` and all other activities to the ones `A`, `B` and `D` have in respect to these activities. This way, we actually replace block {`A,B,D`} by block {`A,B,D,E`} in which `E` and {`A,B,D`} constitute different branches of an XOR-block, i.e., we obtain order matrix A'' representing model S'' in Fig. 4.3.

When compared to a process model and its corresponding process structure tree respectively, the order matrix eases structural process changes. Users only

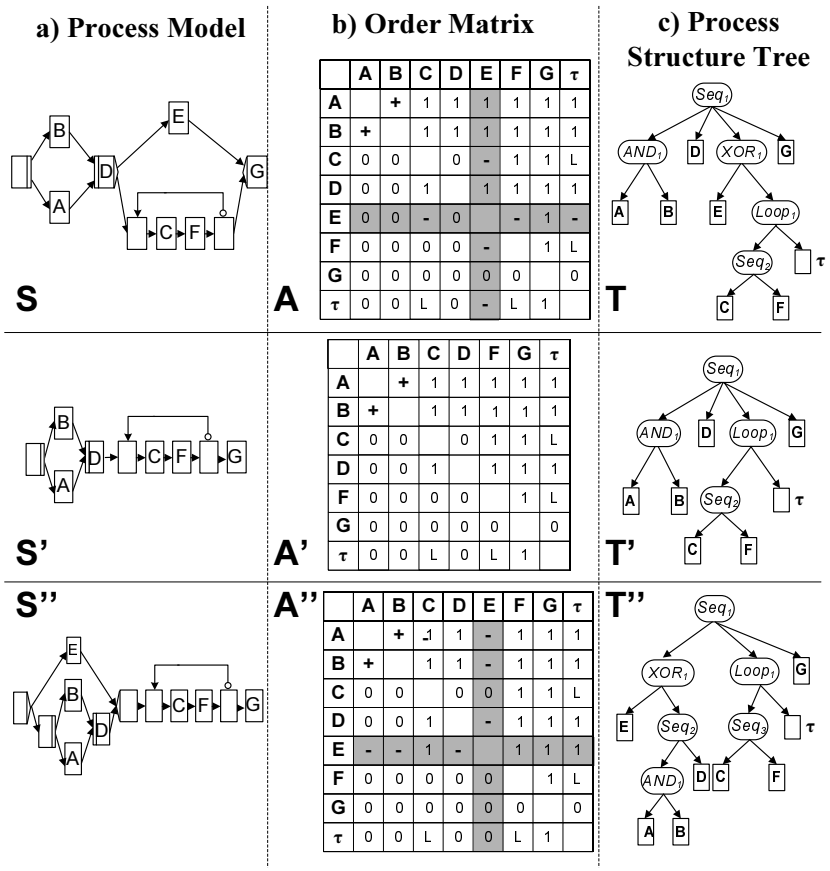


Figure 4.3: Process Model, Order Matrix and Tree

need to focus on the activities to be modified, without worrying about adaptations of the corresponding process model and process tree respectively. For example, moving an activity within a process structure tree often requires modifying the whole tree structure. As illustrated in Fig. 4.3, when moving `E` in S such that `E` and block {`A,B,D`} belong to different branches of an XOR block afterwards (as in S''), almost the whole tree needs to be restructured. Reason is that a process tree strongly depends on the blocks in a process model (in fact, a sub-tree is a block) [204], and changing an activity in a process model often results in changing process blocks as well. However, in the corresponding order matrix we only need to modify the order relations of `E`.

4.3.2 Identifying Process Blocks

One popular reason to transform a process model into a process tree structure is to identify blocks [204, 205, 59, 92]. For example, [205] decomposes a large process model into smaller blocks in order to enable faster validation and verification; [59] allows constructing a process model dynamically using process patterns and process blocks. Order matrices can additionally be used for block analysis in two respects:

1. We can use an order matrix to quickly decide on whether or not certain activities can form a block. Two activities can form a block iff they have the same order relations with all remaining activities. Generally, we can decide whether or not an arbitrary set of activities may form a block. If certain activities have the same order relations to all remaining activities in a process model, they can form a block. As example consider order matrix A in Fig. 4.3: `A,B` and `D` can form a block since they have same order relations with `C,E,F,`τ and `G`; however, this does not apply to `A,B,D` and `E` since activity `E` shows a different order relation to `C` than `A` does. Such judgment has linear complexity.
2. An order matrix helps enumerating all possible blocks within a process model. Based on the aforementioned analysis, we can conclude that two disjoint blocks which have the same order relations to all activities outside these two blocks, may form a larger block. Based on this, a block containing i activities can be identified by judging whether or not two disjoint blocks containing j and k activities (with $i = j+k$) may form a block. Starting with blocks which comprise exactly one activity (i.e., the activities themselves), we can iteratively find all blocks in a process model (Chapter 7 formally describes an algorithm). As example consider order matrix A in Fig. 4.3. All blocks in process model S are listed in Table 4.1.
3. As discussed in Section 2.1.2, a sub-tree within a process structure tree corresponds to a block of its corresponding process model, but NOT vise versa. As example consider Fig. 4.4. Process structure trees T, T_1 and T_2 constitute different trees but represent the same process model (cf. S in Fig.

Block Size	Blocks
1	{A}, {B}, {C}, {D}, {E}, {F}, {G}, {τ}
2	{A,B}, {C,F}
3	{A,B,D}, {C,F,τ}
4	{C,F,τ,E}
5	{C,F,τ,E,D}, {C,F,τ,E,G}
6	{C,F,τ,E,D,G}
7	{A,B,D,C,F,τ,E}
8	{A,B,D,C,F,τ,E,G}

Table 4.1: All blocks of process model S from Fig. 4.3

4.3). In this case, block {A,B,D} corresponds to a sub-tree within T_1, but not a sub-tree in T or T_2. Therefore, identifying all sub-trees in a particular process structure tree does not imply that we have enumerated all blocks in the corresponding process model. Reason is the process structure tree of a process model is not necessarily unique, we refer [204] for the details.

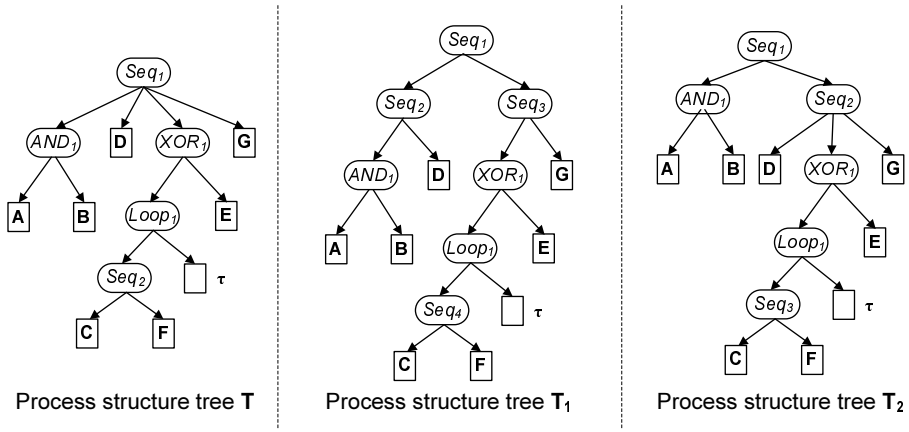


Figure 4.4: Three different process trees representing the same process model

Clearly, for the aforementioned use cases (i.e., judging whether activities can form a block, or enumerating all blocks in a process model) it would be difficult to realize them based on the process structure tree. First, the process structure tree of a process model is not necessarily unique; i.e., the model can be expressed by different tree structures and hence implies different blocks [204]. Second, though there are techniques to decompose process models into tree representations, we are not aware of any technique that can decide in linear time whether certain activities can form a block or that enumerates all blocks of a process model.

4.4 Summary

We provided a matrix representation for block-structured process models, which we denote as order matrix. Such matrix constitutes a unique representation of a process structure tree and consequently a unique representation of a block-structured process model. We compared our order matrix with process tree representations. Results indicate that the order matrix better supports process changes, and fosters the identification of process blocks. In the following chapters, we will discuss some use cases for the order matrix in the context of process changes. This includes an approach for measuring the distance between two block-structured process models, as well as algorithms for mining process variants. In the latter context, we will also provide an intuitive approach for aggregating the order matrices of a process variant collection, as well as for processing such an aggregated order matrix in the context of process variant mining.

5

Measuring Process Model Similarity based on High-level Change Operations

5.1 Introduction

As discussed in Chapter 2, the pivotal research on process flexibility over the last years has provided the foundation for enabling dynamic process changes [187, 11, 141, 224, 87]. Process flexibility denotes the capability of a process-aware information system to reflect externally triggered changes by modifying only those aspects of a process, and its implementation respectively, that need to be changed, while keeping the other parts stable; i.e., the ability to change or evolve the process without completely replacing it [141]. To compare two process models is a fundamental task in this context. In particular, it becomes necessary to calculate the minimal difference between two process models based on high level changes. If we need to transform one model into another, for example, efforts can then be reduced and the transformation can go smoothly; i.e. we do not need to re-define the new process model from scratch, but only apply these high-level changes (cf. Def. 5) either at process type or process instance level (see [154] for an overview of existing techniques for dynamic process changes).

In this chapter, we deal with our second research question (cf. Section 1.2):

How shall we measure the distance between process models such that it can reflect minimal efforts for process model configurations?

Clearly, our focus is on minimizing the number of high-level change operations needed to transform one block-structured process model S into another block-structured process model S'. Soundness of the resulting process model should be also not sacrificed. We apply the high-level change operations as described in Section 2.2 in the given context. By considering high-level changes, we can distinguish our approach from traditional similarity measures like graph or sub-graph isomorphism [181, 235, 235, 166, 86]. Both only consider basic change primitives like the insertion or deletion of single nodes and edges.

Answering the above research question will lead to better cost efficiency when re-designing process models, since the efforts to implement the corresponding changes in the supporting PAIS are minimized. At the process instance level,

this way we can reduce the efforts to propagate process type changes to the running instances [154, 147]. Finally, the derived differences between original process model and its process instances can be used as a set of pure and concise change logs (cf. Def. 6) for process change mining [62].

In Chapter 2, we have provided the technical foundation for users to flexibly change process models at both the process type and the process instance level. For example, users may dynamically *insert, delete* or *move* an activity at both levels [141]. In addition, snapshot differential algorithms [93, 30], as known from database technology, can be used as a fast and secure method to detect the *change primitives* (e.g. to add or delete nodes and edges) needed to transform one process model into the other.

Using the ADEPT change framework and snapshot differential algorithm, this chapter applies Digital Logic in Boolean Algebra [23] to provide a new method to transform a process model into another one based on high-level change operations. This method does not only minimize the number of changes needed in this context, but also guarantees soundness of the changed process model afterwards, i.e., the process model remains correct when applying high-level change operations.[1] We further provide two measures – *process distance* and *process similarity* – based on high-level change operations, which indicate how costly it is to transform process model S into model S', and how different S and S' are.

The remainder of this chapter is organized as follows: Section 5.2 first provides a general description of our approach. Section 5.3 describes required insert and delete operations for transforming a process model into another, while Section 5.4 computes the number of move operation. Finally Section 5.5 concludes with a summary.

5.2 General Description of our Comparison Method

We first introduce our method to detect the minimal number of change operations needed to transform a given process model S into another model S'. As example, consider the process models S and S' in Fig. 5.1. As mentioned in Section 5.1, the key issue is to minimize the number of change operations needed to transform a process model $S \in \mathcal{P}$ into another model $S' \in \mathcal{P}$. Let $T = (N, C, CT, E, l)$ and $T' = (N', C', CT', E', l')$ be the corresponding process structure tree of S and S'. Generally, three steps are needed (cf. Fig. 5.1) to realize this minimal transformation:

1. $\forall n_i \in N \setminus N'$: *delete* all nodes that are present in T, but are not contained in T'. This first step transforms the corresponding process models S to S_{same} (cf. Fig. 5.1b).
2. $\forall n_i \in N \bigcap N'$: *move* all nodes that are contained in both process structure trees to the locations as reflected by T'. Regarding our example, this second

[1] Note that this can be ensured for the control flow perspective. However, to also ensure correction of data flow or other process aspects, additional checks become necessary.

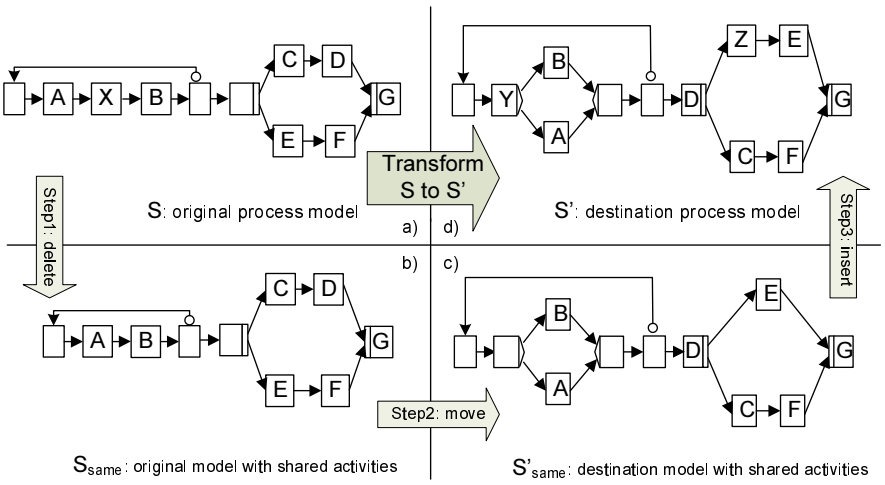


Figure 5.1: Three Steps to Transform S into S'

step transforms the corresponding process model S_{same} to S'_{same} (cf. Fig. 5.1c).

3. $\forall n_i \in N' \setminus N$: *insert* those activities which are contained in T', but not in T. As depicted in Fig. 5.1, the third step transforms the corresponding process model S'_{same} to S' (cf. Fig. 5.1d).

Note that the aforementioned three steps were described based the node sets N and N' in the corresponding process structure tree. Reason is that we want to ignore the silent activities which were often introduced during process transformations (cf. Chapter 4). Insertions and deletions deal with changes of a set of activities. Here, we can hardly do anything to reduce efforts (i.e., to reduce the number of required insert operation and delete operations respectively): New activities ($a_i \in N' \setminus N$) must be added and obsolete activities ($a_j \in N \setminus N'$) be deleted.

The focus of minimality can therefore be shifted to the use of the *move* operation, which changes the structure of a process model, but not its set of activities. Since a move operation logically corresponds to a delete operation followed by an insert operation, we can transform S_{same} to S'_{same} by maximally applying $n = |N \bigcap N'|$ move operations. Reason is that n move operations correspond to deleting all activities and then re-inserting them at their new positions. Correspondingly, n is the maximum number of change operations needed to transform one process model into another, both with same set of activities (S_{same} and S'_{same} in our example from Fig. 5.1). To measure the complete transformation from S to S', we formally define *process distance, bias* and *process similarity* as follows:

Definition 9 (Process Distance, Bias and Similarity)
Let $S, S' \in \mathcal{P}$ be two process models, and let $T = (N, C, CT, E, l)$, $T' =$

(N', C', CT', E', l) be their corresponding process structure trees. Let further $\sigma = \langle \Delta_1, \Delta_2, \ldots \Delta_n \rangle \in \mathcal{C}^*$ be a sequence of change operations transforming S into S' (i.e. $S[\sigma\rangle S'$).

1. ***Distance*** *$d_{(S,S')}$ between S and S' corresponds to the minimal number of high-level change operations needed to transform S into S'; i.e., we define*

$$d_{(S,S')} = \min\{|\sigma| \mid \sigma \in \mathcal{C}^* \wedge S[\sigma\rangle S'\} \quad (5.1)$$

2. *A sequence of change operations σ with $S[\sigma\rangle S'$ and $|\sigma| = d_{(S,S')}$ is denoted as a **bias** between S and S'.*

3. *At last, let $m = |N| + |N'| - |N \bigcap N'|$ be the maximal number of change operations needed to transform S into S'. Then process **similarity** between S and S' equals to $\frac{m - d_{(S,S')}}{m}$, i.e., similarity equals to ((maximal number of changes - minimal number of changes) / maximal number of changes).*

Generally, it is possible to have more than one minimal sequence of change operations to transform S into S', i.e., given process models S and S' their bias does not need to be unique. A detailed discussion of this issue can be found in [191, 187].

5.3 Determining Required Activity Deletions and Insertions

To accomplish Step 1 and Step 3 of our approach (cf. Section 5.2), we have to deal with the change of the activity set when transforming S into S'. It can be easily detected by applying existing snapshot algorithms [93, 30] to both S and S'. As described in Section 5.2, as first step we need to *delete* all activities $a_i \in N \setminus N'$ that are contained in S, but not in S'. Regarding our example from Fig. 5.1, we can derive as our first high-level change operation $\Delta_1 = delete(S, \mathtt{X})$. Similarly, activities that are contained in S', but not in S, are inserted in Step 3, after having moved the commonly shared activities to their respective position in S' (S'_{same} respectively). The parameters of the insert operation, i.e., the predecessors and successors of the inserted activity can be identified in S'. In this way, we obtain the last two change operations for our example: $Insert(S, \mathtt{Y}, \mathtt{StartLoop}, \{\mathtt{A}, \mathtt{B}\})$ and $Insert(S, \mathtt{Z}, \mathtt{D}, \mathtt{E})$.

5.4 Determining Required Move Operations

We now focus on Step 2 of our method; i.e., to transform two process models with same activity set by applying move operations. Here, we can ignore the activities which are not contained in both models, i.e., in S and S' (cf. 5.3). Instead, we

consider the two process models S_{same} and S'_{same}, respectively, as depicted in Fig. 5.1.

We revisit our example from Fig. 5.1. The order matrices of S_{same} and S'_{same} are depicted in Fig. 5.2. Note that both S_{same} and S'_{same} contain a silent activity τ, which represents the loop structure (cf. Def. 8). When comparing two process models, it is sufficient to compare their order matrices (cf. Def. 8), since an order matrix can uniquely represent the process model (cf. Chapter 4). This also means that the *differences of two process models* can be related to the *differences of their order matrices*. If two activities have different execution order in two process models with same activity set, a *conflict* can be defined as follows:

Definition 10 (Conflict) *Let $S, S' \in \mathcal{P}$ be two sound and block-structured process models and. Let further $T = (N, C, CT, E, l)$, $T' = (N', C', CT', E', l')$ and A, A' be the corresponding process structure trees and order matrices for S and S' respectively. In this context, we have $N = N'$. Then: Activities a_i and a_j are* conflicting *iff $A_{ij} \neq A'_{ij}$. We formally denote this as $C_{(a_i,a_j)}$. $\mathcal{CF} := \{C_{(a_i,a_j)} \mid a_i, a_j \in N \wedge A_{ij} \neq A'_{ij}\}$ then corresponds to the set of all conflicts existing between S and S'.*

Fig. 5.2 marks up differences between the two order matrices in grey. The set of conflicts is as follows: $\mathcal{CF} = \{C_{(\mathtt{A},\mathtt{B})}, C_{(\mathtt{C},\mathtt{D})}, C_{(\mathtt{C},\mathtt{F})}, C_{(\mathtt{D},\mathtt{E})}, C_{(\mathtt{D},\mathtt{F})}, C_{(\mathtt{E},\mathtt{F})}\}$.

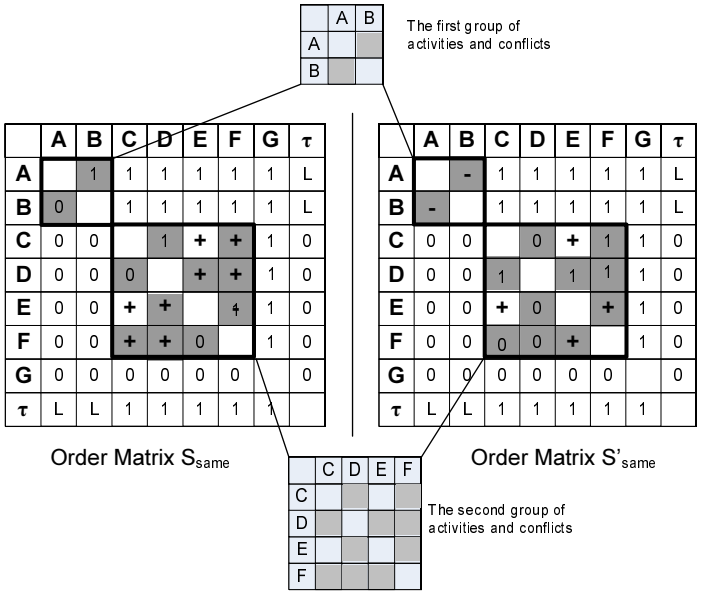


Figure 5.2: Order Matrices of S_{same} and S'_{same} from Fig. 5.1

5.4.1 Optimizing the Conflicts

To evolve S_{same} into S'_{same} (cf. Fig. 5.1), we have to eliminate conflicts between these two models by applying *move* operations. Obviously, if there is no conflict for the two models, they will be identical. Every time we move an activity from its current position in S_{same} to the position it has in S'_{same} we can eliminate the conflicts this activity has with other activities. For example, consider activity `A` in Fig. 5.1. If we move `A` from its position in S_{same} (preceding `B`) to its new position in S'_{same} (`A` and `B` are contained in two different branches of a conditional branching block) we can eliminate conflict $C_{(\texttt{A},\texttt{B})}$.

As shown in the order matrices, moving `A` requires two steps. First, we have to set the elements in the first row and first column of $A_{n\times n}$ (which corresponds to activity `A`) to empty, since `A` is moved away. Second, we have to reset these elements according to the new order relation of `A` in comparison to the other activities from S'_{same}. So every time we move an activity, we are able to change the value of its corresponding row and column in the order matrices, i.e., we change these values corresponding to the original model to the values being compliant with the target model. By doing this iteratively, we can change all the values and eliminate all the conflicts so that we finally achieve the transformation from S_{same} to S'_{same}.

A non-optimal solution would be to move all the activities involved in the conflicts, as set out by $\mathcal{CF}$, from their positions in S_{same} to the positions they have in S'_{same}. Regarding our example from Fig. 5.2, in order to apply this straightforward method, we would need to move activities `A`, `B`, `C`, `D`, `E` and `F` from their positions in S_{same} to the ones in S'_{same}. However, this naive method is not in line with our goal to minimize the number of applied change operations. For example, after moving activity `A` from its current position in S_{same} to the position it has in S'_{same}, we do not need to move activity `B` anymore. Note that after applying this change operation, there are no activities with which activity `B` still has conflicts.

Digital logic in Boolean algebra [23] helps to solve this minimization problem. Digital logic constitutes the basis for digital electronic circuit design and optimization. In this field, engineers face the challenge to optimize the internal circuit design given the required input and output signals. To apply such technique in our context, we consider each process activity as an independent input signal. We want to design a circuit which can cover all conflicts defined by $\mathcal{CF}$ (cf. Def 10). If activity a_i conflicts to activity a_j, we can either move one of them or both of them from the positions they have in S_{same} to the ones they have in S'_{same}. Doing so, the conflict will not exist anymore. Reason is that every time we move an activity from the position it has in S_{same} to the position it has in S'_{same}, we reset the corresponding row and column of this activity in the order matrix. A conflict can be interpreted as a digital signal: When the two input signals a_i and a_j are both "true"(this means we do not move activities a_i and a_j), we cannot solve the conflict and the 'circuit' shall give an output signal of "false". If we apply this to all conflicts in $\mathcal{CF}$, we will obtain all "false" signals. Meanwhile,

the "circuit" should be able to tell us what will result in a "true" output (i.e., the negative of all "false" signals). This "true" output expresses which activities we need to move. Regarding our example from Fig. 5.2, given the set of conflicts $\mathcal{CF}$, our logic expression then is as follows: $\overline{\texttt{AB} + \texttt{CD} + \texttt{CF} + \texttt{DE} + \texttt{DF} + \texttt{EF}}$.

The complexity for optimizing the logic expression is $\mathcal{NP}$-*hard* [23]. Therefore it is advantageous to reduce the size of the problem. Concerning our example, we can cut down the optimization problem into two groups: one with activities A and B, as well as conflict $C_{(\texttt{A},\texttt{B})}$; another one with activities C, D, E and F, as well as the following set of conflicts: $\{C_{(\texttt{C},\texttt{D})}, C_{(\texttt{C},\texttt{F})}, C_{(\texttt{D},\texttt{E})}, C_{(\texttt{D},\texttt{F})}, C_{(\texttt{E},\texttt{F})}\}$. Such division can be achieved in $O(n)$ time by applying the following three steps.

- Step 1: List all conflicting activities, and consider each activity as an independent group.
- Step 2: If conflicting activities a_i and a_j (i.e., $C_{(a_i,a_j)}$) are contained in two different groups, merge these two groups.
- Step 3: Repeat Step 2 for all conflicts in $\mathcal{CF}$.

After applying these three steps, we can divide the activities as well as the associated conflicts into several groups. Regarding our example, the optimization problem can be divided into two sub-optimization problems: $\overline{\texttt{AB}}$ and $\overline{\texttt{CD} + \texttt{CF} + \texttt{DE} + \texttt{DF} + \texttt{EF}}$. We depict this by the two small matrices in Fig. 5.2.

Optimizing logic expressions has been intensively discussed in Discrete Mathematics. Therefore we omit details here and refer to Karnaugh map [23, 120] and Quine-McCluskey algorithm [23, 36]. We have implemented the latter in our proof-of-concept prototype. Regarding our example in Fig. 5.1, the two optimization results are $\overline{\texttt{AB}} = \bar{\texttt{A}} + \bar{\texttt{B}}$ for the first group and $\overline{\texttt{CD} + \texttt{CF} + \texttt{DE} + \texttt{DF} + \texttt{EF}} = \bar{\texttt{D}}\bar{\texttt{F}} + \bar{\texttt{C}}\bar{\texttt{E}}\bar{\texttt{F}} + \bar{\texttt{C}}\bar{\texttt{D}}\bar{\texttt{E}}$ for the second group. We can interpret this result as follows. For the second group, either we move activities D and F, or we move activities C, E and F, or we move activities C, D and E from their position in S_{same} to the positions they have in S'_{same}. Based on this we can transform S_{same} into S'_{same} since all conflicts are eliminated. As can be seen from the order matrices, if we change the value of the corresponding rows and columns of these activities in S_{same}, we can turn S_{same} into S'_{same}. Since we want to minimize the number of change operations, we can draw the conclusion that activities D and F must be moved. Same rule applies to the result of the first group. However, there is no difference whether to move either A or B since both operations count as one change operation. Here, we arbitrarily decide to move activity B.

So far we have determined the set of activities to be moved. The next step is to determine the positions where these activities need to be moved to. Operation $move(S, \texttt{X}, \mathcal{A}, \mathcal{B}, [sc])$ will be independent from other move operations (i.e., it does not matter in which order to move the respective activity) if its direct predecessors $\mathcal{A}$ and direct successors $\mathcal{B}$ do not belong to the set of activities to be moved. Regarding our example from Fig. 5.1, activity F satisfies this condition since its predecessor C and its successor G are not moved. Same results applies

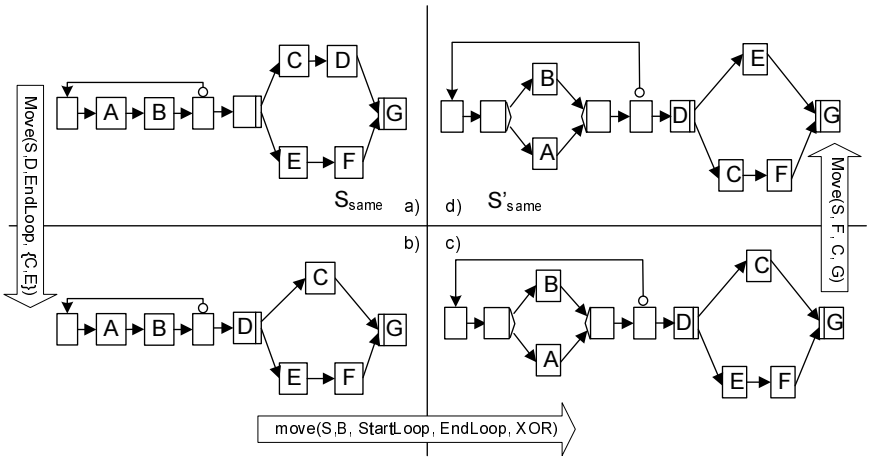


Figure 5.3: Process models after every move operation

to D while its predecessor EndLoop and successors C and E are not changed. Note that since activity B is moved to a position where it forms an XOR block with activity A, two silent activities denoting the XOR-split and XOR-join have been automatically inserted. A detailed discussion can be found in [141].

According to the position the moved activities have in S'_{same}, we can determine the parameters (i.e., the predecessors, successors and conditions) for every move operation. In S'_{same}, activity D has predecessors EndLoop, and successors E and C. So one move operations therefore is $move(S, \texttt{D, EndLoop,\{C,E\}})$. Similarly, we obtain the other two move operations: $move(S, \texttt{B,EndLoop,StartLoop}, \textit{[sc]})$ and $move(S, \texttt{F,C,G})$. The intermediate process models resulting after every move operation are shown in Fig. 5.3. When comparing order matrices for each model in Fig. 5.3, it becomes clear that every move operation changes the values of the row and the column corresponding to the moved activity.

5.4.2 Distance and Similarity between other Models

The method described in Section 5.4 has been tested and implemented in the ADEPT framework, and will be used in the following chapters. Taking our example from Fig. 5.1 (i.e., to transform S into S'), the following *six* change operations are required: $\sigma = \{delete(S, \texttt{X}), move(S, \texttt{F,C,G}), move(S, \texttt{D,\{A,B\},\{C;E\}}), move(S, \texttt{B,StartFlow,D}, XOR), insert(S, \texttt{Y,StartFlow,\{A,B\}}),$ and $insert(S, \texttt{Z,D,E})$ }. Distance between the two models is *six*, bias is σ and similarity is *0.4* (cf. Def. 9). To illustrate our method and these numbers in more detail, we compare the distances and similarities between the seven process models discussed so far: S, S_1 and S_2 from Fig. 3.2 and S, S_{same}, S'_{same} and S' from Fig. 5.1. Distance and similarity of two models are specified as *distance/similarity* in each corresponding cell in Fig. 5.1. As the transforma-

tion is commutable, we only fill in the upper triangle matrix. Taking Fig. 5.1, we can conclude:

		Figure 5.1				Figure 5.3	
		S	S_{same}	S'_{same}	S'	S_{I1}	S_{I2}
Figure 5.1	S	0 / 100%	1 / 89%	4 / 56%	6 / 45%	2 / 78%	3 / 67%
	S_{same}		0 / 100%	3 / 63%	5 / 50%	1 / 88%	2/ 75%
	S'_{same}			0/ 100%	2 / 80%	2 / 75%	1/ 88%
	S'				0 / 100%	4 / 60%	3/ 70%
Figure 5.3	S_{I1}					0 / 100%	1 / 88%
	S_{I2}						0 / 100%

Table 5.1: Distance and similarities of different process models

1. Changing the activity set always leads to a modified distance. For example, $d_{(S_n,S'_{same})}$ always equals $d_{(S_n,S')} + 2$, where S_n stands for a process model other than S' or S'_{same} in Fig. 5.1. Reason is that S' contains two unique activities `Y` and `Z` when compared to S'_{same}, while the rest are identical.

2. If three process models S, S', and S'' have same activity sets, we will obtain $d_{(S,S'')} \leq d_{(S,S')} + d_{(S',S'')}$. It is easy to understand this because some activities could be moved twice when transforming S into S' and S' into S''.

5.5 Summary

We provided a method to quantitatively measure the distance and similarity between two process models based on the efforts for model transformation. High-level change operations are used to evaluate the similarity since they guarantee soundness and also provide more meaningful results. We further applied digital logic known from Boolean algebra such that the number of change operations required to transform process model S into process model S' becomes minimal.

In the following chapters, we will apply respective distance and similarity measures in the context of algorithms we developed for process variant mining.

6

Mining Process Variants Using a Clustering Technique

6.1 Introduction

Though considerable efforts have been made to ease process configuration and adaptation [66, 141, 165], we have not utilized the knowledge resulting from these process model changes yet [213]. Fig. 6.1 describes the goal of this chapter as it has been already motivated in Chapter 1. We aim at learning from past process changes by "merging" existing process variants into one generic process model, which "covers" these variants best. Thereby we do not presume any knowledge of the original reference model the variants were derived from. By adopting this generic model as *reference process model* within the PAIS, cost of change and need for future process adaptations will decrease. Chapter 6 deals with our third research question (cf. Section 1.2):

Given a collection of process variants, how can we discover a reference process model in such a way that average distance between it and the process variants becomes minimal?

The distance between reference process model and process variant is measured in terms of the number of high-level change operations (e.g., to insert, delete or move activities [141]) needed to transform the reference model into the respective variant model (cf. Def. 9). *Change distance* directly represents the efforts needed for process adaptation and customization, and *average change distance* between a reference model and a collection of process variants directly measures the configuration efforts for a particular reference process model. Obviously, the challenge is to find the "best" reference model, i.e., the one with minimal average distance to the known variants. Note that we only need a collection of process variants as input of our analysis. In particular, we do not require a change log (cf. Def. 6) as input, which specifically documents all change operations performed during the configuration of process variants [61, 62]. In fact, even the original reference process model from which the variants were derived is not really required. In the following we present a clustering technique to deal with these challenges.

The remainder of this chapter is organized as follows. Section 6.3 presents

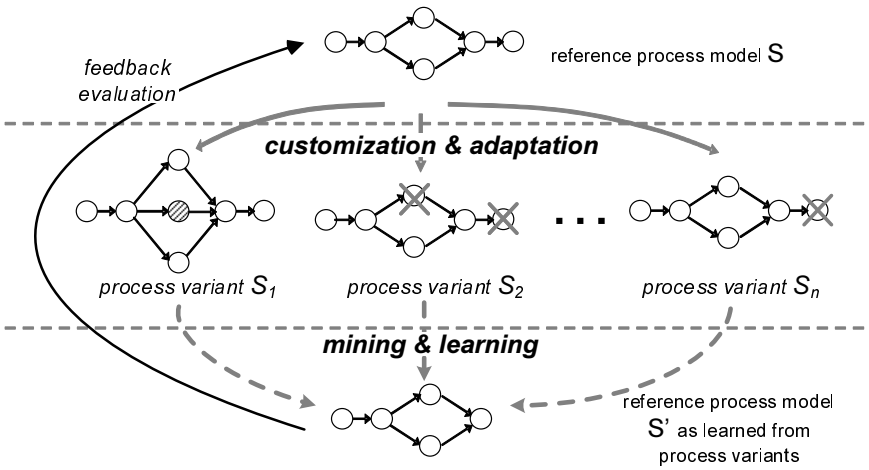


Figure 6.1: Mining a new reference model

our basic clustering algorithm for mining process variant models having same activity set. Section 6.4 extends it such that variants with different activity sets can be considered as well. We apply simulations to systematically evaluate the performance of our algorithm in Section 6.5. Finally, we conclude with a summary in Section 6.6.[1]

6.2 Illustrative Example

To illustrate our mining approach, Fig. 6.2 depicts an example comprising six different process variants $S_i \in \mathcal{P}$ $(i = 1, 2, \ldots 6)$ and being based on common workflow patterns like AND-split, AND-join, XOR-split, XOR-join, and Loop [193]. Note that these variants do not only differ in structure, but also in respect to their activity sets. For example, activity `X` appears in 5 out of the 6 variants (except S_2), while activity `Z` only appears in S_5. Furthermore the six variants are weighted. In the context of our work, we define the **weight** w_i of a process variant S_i as the number of process instances that were executed on basis of S_i. In our example, 25 instances were executed on basis of process variant S_1, while 20 instances ran on S_2. If we only know the process variants, but have no runtime information about related instance executions, we could assume variants to be equally weighted; i.e., every process variant will then have weight 1.[2]

[1] A case study in which we applied our algorithm in practice is presented in Chapter 9.

[2] Note that in this context, the weight w_i of a variant S_i only reflects the frequency with which S_i was executed. We do not consider which path was taken when executing a process model, or in which order activities were executed in each process instance (cf. Section 2.1.3). Note that this is different to process mining techniques (cf. Section 2.5) which are often based on trace analyses (cf. Def. 4).

In Section 6.3 we first assume that all process variants have same activity sets, i.e., we focus on activities `A,B,C,D,E,F,G,H,I` and `J`, which exist in all six process variants S_1 - S_6, but have different order relations. In Section 6.4, we relax this constraint and extend our algorithm to deal with activities that do not appear in all process variants as well.

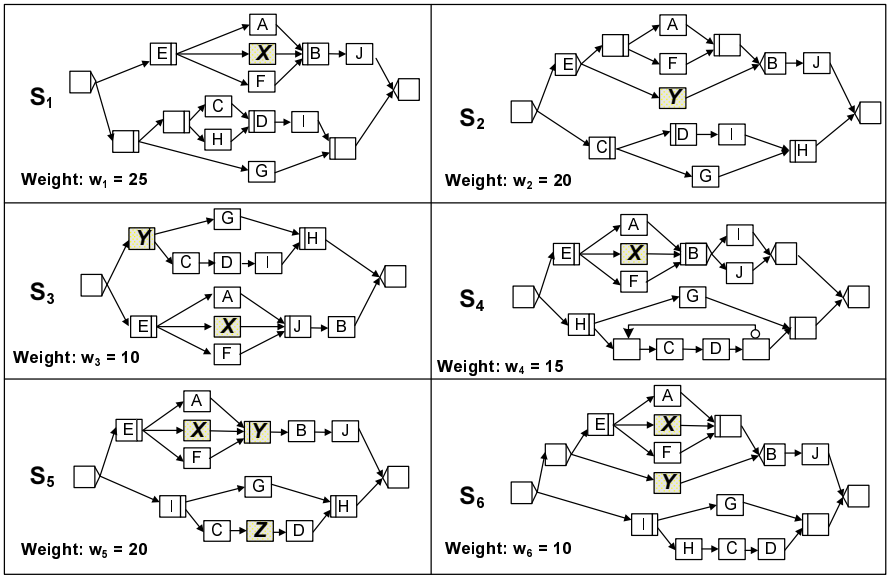


Figure 6.2: Illustrative example

6.3 Clustering Approach for Discovering Reference Process Models

We now present a *clustering-based algorithm* for mining a collection of process variants. Our goal is to derive a new reference model out of a given collection of process variants which is easier configurable than the current one. Since we restrict ourselves to block-structured process models, we can build the new reference model by enlarging blocks, i.e., we first identify two activities that can form a block, then we merge this block with other activities and blocks respectively to form a larger block, and so forth. This procedure continues until all activities and blocks respectively are merged into one single block. This block and its internal structure then represent the new reference process model we are looking for.

Basically, our clustering approach for mining process variants works as follows:

1. For all process variants calculate their order matrices (cf. chapter 4). Aggregate them to one high-dimensional matrix representing all variants (cf.

Section 6.3.1).

2. Based on this high-dimensional matrix, determine activities to be clustered in a block (cf. Section 6.3.2).

3. Determine the order relation the clustered activities shall have within this block (cf. Section 6.3.3).

4. After building a new block in Steps 2 and 3, reflect on the clustering of the activities by adjusting the high-dimensional matrix accordingly (cf. Section 6.3.4).

5. Repeat Steps 2, 3 and 4 until all activities are clustered together; i.e., until the new process model has been constructed by enlarging blocks.

6.3.1 Representing a Collection of Process Variants as Aggregated Order Matrix

For each variant of the given collection of process variants, we first compute its order matrix (cf. Def. 8 in Chapter 4). Regarding our example from Fig. 6.2, we need to determine six order matrices, which are shown on top of Fig. 6.3. Afterwards, we analyze the order relation for each pair of activities considering all order matrices derived before. In this context, we consider activities from different variants are the same if they have the same label.[3] As the order relation between two activities might not always be the same in all order matrices, this analysis does not result in a fixed relationship, but provides a distribution for the five types of order relations (cf. Def. 8). Regarding our example, for instance, in 50% of all cases activity `H` is a successor of activity `C` (as in S_2, S_3 and S_5), in 25% of all cases `H` precedes `C` (as in S_4 and S_6), and in 25% of a cases `H` and `C` are contained in different branches of an AND block (as in S_1) (cf. Fig. 6.3). Generally, we can define the order relation between two activities `a` and `b` as 5-dimensional vector $V_{ab} = (v^0_{ab}, v^1_{ab}, v^+_{ab}, v^-_{ab}, v^L_{ab})$. Each field then corresponds to the frequency of the respective relation type ('0', '1', '+', '-' or 'L') as specified in Def. 8.

Take again our running example and consider Fig. 6.3. Here, $v^0_{\mathtt{HC}}$ corresponds to the frequency of all cases with activities `H` and `C` having order relationship '0', i.e., all cases for which `H` succeeds `C`; we obtain $V_{OQ} = (0.5, 0.25, 0.25, 0, 0)$.

Formally, we define an *aggregated order matrix* as follows:

[3] Otherwise, we refer to [44] for an approach that matches activities from different process models in case they have different labels. Note that we can also use this technique to handle silent activities which represent the loop structures in a process structure tree. When there are multiple silent activities in each of the process structure trees, we can map these silent activities based on their context (e.g., their relationship to other activities). In the following, we assume that such mapping between activities (including silent ones) in different process structure trees has already been established and we can map them simply based on their labels.

Definition 11 (Aggregated Order Matrix) *Let $S_i \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants. Let further $T_i = (N_i, C_i, CT_i, E_i, l_i)$ and A_i be the process structure tree and the order matrix of S_i, and w_i be the number of process instances that were executed on S_i. The **Aggregated Order Matrix** of all process variants is defined as 2-dimensional matrix $V_{m \times m}$ with $m = |\bigcup N_i|$ and each matrix element $v_{a_j a_k} = (v^0_{a_j a_k}, v^1_{a_j a_k}, v^+_{a_j a_k}, v^-_{a_j a_k}, v^L_{a_j a_k})$ being a 5-dimensional vector. For $\diamond \in \{0, 1, +, -, L\}$, element $v^\diamond_{a_j a_k}$ expresses to what percentage, activities a_j and a_k have order relation $\diamond$ within the collection of process variants $S_1, \ldots, S_n$. Formally: $\forall a_j, a_k \in \bigcup N_i, a_j \neq a_k$:*

$$v^\diamond_{a_j a_k} = \frac{\sum_{A_{i_{a_j a_k}} =' \diamond'} w_i}{\sum_{a_j, a_k \in N_i} w_i}. \tag{6.1}$$

The aggregated order matrix V of the process variants from Fig. 6.2 is shown in Fig. 6.3. Due to space limitations we only show a partial view of the order matrices here (i.e., activities `A, B, C, F, H, I` and `J`). Generally, the main diagonal of an aggregated order matrix is always empty since we do not specify the order relation of an activity with itself.

In Chapter 4 we have shown that we can transform an order matrix into a process model by identifying blocks: i.e., two activities can be clustered into a block if they have same order relation with respect to other activities. As we will show, a similar idea can be applied when analyzing an aggregated order matrix. Our goal is to derive an optimal reference process model for the given variants based on this representation form.

6.3.2 Determining the Activities to be Clustered

This subsection describes how we derive the blocks for the reference model to be discovered from an aggregated order matrix, i.e., from a collection of process variants. There are two fundamental issues we have to consider in this context. First, we have to decide which activities (and blocks respectively) shall be "blocked". Second, we must choose an order relation for them. This subsection deals with the first issue, the second one is addressed in Section 6.3.3.

Regarding an order matrix two activities can be clustered in a block if they have same order relations with respect to the other activities (cf. Chapter 4). We can apply a similar idea when analyzing an aggregated order matrix. However, the relationship between two activities in an aggregated order matrix is expressed as 5-dimensional vector showing the distribution of the order relations over all process variants. When determining pairs of activities that can be clustered in a block, it would be too restrictive to require precise matching as in the case of an order matrix. To deal with this, we first introduce function $f(\alpha, \beta)$ which expresses the closeness between two vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$:

$$f(\alpha, \beta) = \frac{\alpha \cdot \beta}{|\alpha| \times |\beta|} = \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2} \times \sqrt{\sum_{i=1}^{n} y_i^2}} \tag{6.2}$$

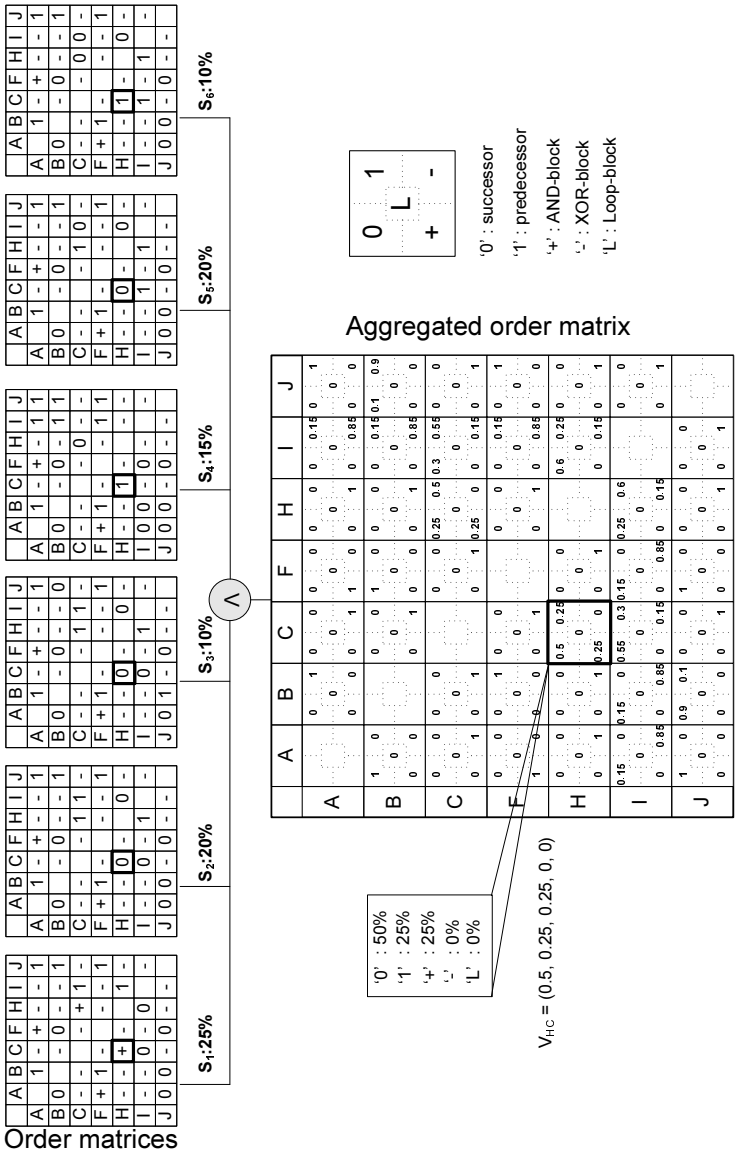


Figure 6.3: Aggregated order matrix V

$f(\alpha, \beta) \in [0, 1]$ computes the cosine value of the angle θ between vectors α and β in Euclid space. If $f(\alpha, \beta) = 1$ holds, α and β exactly match in their directions; $f(\alpha, \beta) = 0$ means, they do not match at all. When comparing closeness between $v_{\mathtt{HC}} = (0.5, 0.25, 0.25, 0, 0)$ and $v_{\mathtt{IC}} = (0.55, 0.3, 0, 0.15, 0)$, for example, we obtain $f(v_{\mathtt{HC}}, v_{\mathtt{IC}}) = 0.887$. This high value implies that the two vectors are close to each

other though they are not the same.

Using $f(\alpha, \beta)$ we introduce the **Separation** metrics. It indicates to what degree two activities of an aggregated order matrix are suited for being clustered in a block. More precisely, $Separation(a, b)$ expresses how similar order relations of activities a and b are when compared to the other activities. In our example from Fig. 6.2, $Separation(\texttt{A,B})$ is determined by the closeness (measured in terms of the cosine value) of $f(v_{\texttt{AC}}, v_{\texttt{BC}})$, $f(v_{\texttt{AD}}, v_{\texttt{BD}})$, $f(v_{\texttt{AE}}, v_{\texttt{BE}})$, ..., $f(v_{\texttt{AI}}, v_{\texttt{BI}})$ and $f(v_{\texttt{AJ}}, v_{\texttt{BJ}})$. Generally, we define cluster separation as follows:

$$Separation(a, b) = \frac{\sum_{x \in N \setminus \{a,b\}} f^2(v_{ax}, v_{bx})}{|N| - 2} \tag{6.3}$$

N corresponds to the set of activities. Like most clustering algorithms [181], we square the cosine value to emphasize the differences between the two compared vectors. Finally, dividing this expression by $|N| - 2$ normalizes its value to a range between [0, 1]. Regarding our example from Fig. 6.2, we obtain $Separation(\texttt{A,B}) = 0.776$. This separation value indicates that activities `A` and `B` has relatively high similarity regarding their order relations to the remaining activities (relatively high chance of forming same block).

We determine the pair of activities best suited to form a block by measuring how much each activity pair is separated from the other activities. We accomplish this by computing the separation value for each activity pair. The higher this value is, the better the two activities are suited for being clustered. Fig. 6.4 depicts the separation values for our running example from Fig. 6.2. We denote this table as *separation table*. Obviously, activities `A` and `F` have the highest separation value of 1 (marked up in grey color in Fig. 6.4). We therefore choose `A` and `F` as the activities forming our first block.[4] Since $Separation(\texttt{A,F}) = 1$ holds, `A` and `F` can form a block also in all six process variants. We obtain same results when directly analyzing the variants (cf. Fig. 6.2).

6.3.3 Determining the Internal Order Relations

After having decided that activities `A` and `F` are clustered in the first block, we have to determine the order relation these two activities shall have. In addition, we measure how good our choice is. For this purpose, we introduce **Cohesion** as measure which indicates how significant particular order relations between two activities of the same cluster are.

In the aggregated order matrix of our running example, the relationship between activities `A` and `F` is depicted as 5-dimensional vector $v_{\texttt{AF}} = (0, 0, 1, 0, 0)$. It shows the distribution values of the five types of order relations. Obviously, when building a reference process model, only one of the five order relations can be chosen. Therefore, we want to choose that type of order relation which

[4] However, when dealing with more complex examples, there can be several maximal separation values. For this case, we compute cohesion values (cf. Section 6.3.3) between the pairs with the highest separation values. As result, the pair with highest cohesion should be selected, since the relationship of the respective two activities (measured by the cohesion) is most significant.

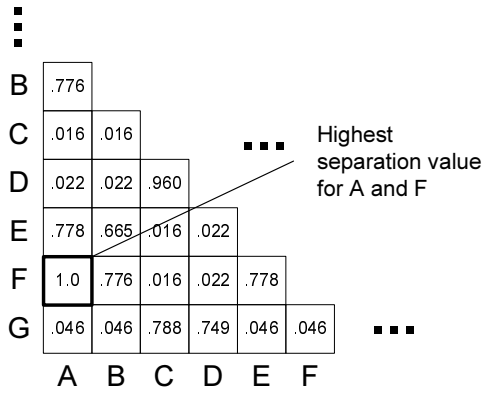


Figure 6.4: Separation table of aggregated order matrix

is most significant. Regarding our example, the significance of each order relation can be evaluated by the closeness vector $v_{\texttt{AF}}$ and the five axes in the 5-dimensional space have. These axes can be represented by five benchmarking vectors: $v^0 = (1,0,0,0,0)$, $v^1 = (0,1,0,0,0)$, $v^+ = (0,0,1,0,0)$, $v^- = (0,0,0,1,0)$, and $v^L = (0,0,0,0,1)$. Based on this, we can compute the significance of each order relation using formula $f(\alpha,\beta)$ (cf. Section 6.3.2), where $\alpha = v_{\texttt{PQ}}$ and β is one of the five benchmarking vectors. Regarding our example, the closest axis to $v_{\texttt{PQ}}$ is v^+ (with $f(v_{\texttt{AF}}, v^+) = 1$). Therefore, we decide that `A` and `F` shall form an AND block (cf. Def. 8).

We use Cohesion to evaluate how good our choice is:

$$Cohesion(a,b) = \frac{\max_{\diamond=\{0,1,+,-,L\}}\{f(v_{ab}, v^{\diamond})\} - 0.4472}{1 - 0.4472} \qquad (6.4)$$

The value range of $\max_{\diamond=\{0,1,+,-,L\}}\{f(v_{ab}, v^{\diamond})\}$ is [0.4472,1]. We use Formula 6.4 to normalize $Cohesion(a,b)$ into value range [0,1]. $Cohesion(a,b)$ equals *one* if there is a dominant order relation, i.e., v_{ab} is on one of the five axes. $Cohesion(a,b)$ equals *zero* if v_{ab} is $(0.2, 0.2, 0.2, 0.2, 0.2)$, i.e., no order relation is more significant than the others. Regarding our example, $Cohesion$(`A,F`) equals 1. This indicates that `A` and `F` have order relation '+' in all six process variants; we can obtain same results by directly analyzing the variants (cf. Fig. 6.2).

6.3.4 Recomputing the Aggregated Order Matrix

We have discovered the first block of our reference process model, which contains `A` and `F`. We have further decided that `A` shall precede `F`, and that the significance of this order relation is 1. We now have to decide on the relationship between the newly created block (comprising `A` and `F`) and the other activities. This requires the adaptation of the original aggregated order matrix in order to represent the

situation in which A and F are clustered in a block.[5] We accomplish this adaptation by computing the means of the order relations between {A, F} and the remaining activities. For example, as $v_{\mathtt{AB}} = (0, 1, 0, 0, 0)$ and $v_{\mathtt{FB}} = (0, 1, 0, 0, 0)$, the order relation between the newly created block (A,F) and activity B corresponds to $(v_{\mathtt{AB}} + v_{\mathtt{FB}})/2 = (0, 1, 0, 0, 0)$.[6] Such computation is applied to all remaining activities outside this block.

Generally, after clustering activities a and b, the new aggregated order matrix V' can be calculated as follows:

$$\forall x \in N \setminus \{a, b\} : \begin{cases} v'_{(a,b)x} = (v_{ax} + v_{bx})/2 \\ v'_{x(a,b)} = (v_{xa} + v_{xb})/2 \end{cases} \quad (6.5)$$

$$\forall x, y \in N \setminus \{a, b\} : v'_{xy} = v_{xy} \quad (6.6)$$

The aggregated order matrix V' we obtain after clustering A and F is shown in Fig. 6.5. Since A and F are replaced by a block $\{A, F\}$ containing these two activities, the matrix resulting after the re-computation is one dimension smaller than V. Afterwards, we treat this block like a single activity, but keep its internal structure in order to build up the new reference process model at the end.

6.3.5 Mining Result

After obtaining the newly aggregated order matrix, we repeat the three steps as described in Sections 6.3.2, 6.3.3 and 6.3.4; i.e., we first identify the two activities (and blocks respectively) to be clustered next, then we determine their order relation within the block, and finally we re-compute the aggregated order matrix considering the newly determined block. In every iteration, we merge two activities and blocks respectively into one bigger block. This iterative clustering continues until all activities from the original aggregated order matrix are clustered. Finally, we obtain our new reference process model. Obviously, the number of required iterations equals the number of activities minus one. Regarding our running example, Fig. 6.6 depicts the final result S' we obtain after running through all iterations of our clustering algorithm.

Fig. 6.6 does not only show process model S', which we have discovered through the mining of the variants from Fig. 6.2, but also the intermediate results we obtain after every iteration (indicated through number at the right-bottom corner of each block). In iteration 1, for example, A and F are clustered to form a block. In the second iteration, this block is merged with activity B as its

[5] Our approach is different to traditional clustering algorithms [181], in which only distances are re-computed, but not the original dataset.

[6] This approach is an unweighted one; i.e., we simple take the average of the two vectors without considering their importance; e.g., how many activities are included in the block. In this way, we can ensure that when merging two blocks of different sizes, the order relations of the resulting block are not too much dominated by the bigger one. Such unweighted approach is widely used in other clustering approaches [181].

	{A,F}	B	C	H	I	J
{A,F}		0 1 0 0 0	0 0 0 0 1	0 0 0 0 1	0 0.15 0 0 0.85	0 1 0 0 0
B	1 0 0 0 0		0 0 0 0 1	0 0 0 0 1	0 0.15 0 0 0.85	0.1 0.9 0 0 0
C	0 0 0 0 1	0 0 0 0 1		0.25 0.5 0 0.25 0	0.3 0.55 0 0 0.15	0 0 0 0 1
H	0 0 0 0 1	0 0 0 0 1	0.5 0.25 0 0.25 0		0.6 0.25 0 0 0.15	0 0 0 0 1
I	0.15 0 0 0 0.85	0.15 0 0 0 0.85	0.55 0.3 0 0 0.15	0.25 0.6 0 0 0.15		0 0 0 0 1
J	1 0 0 0 0	0.9 0.1 0 0 0	0 0 0 0 1	0 0 0 0 1	0 0 0 0 1	

Figure 6.5: Aggregated order matrix V' resulting after the clustering of `A` and `F`

predecessor. Finally, after the nine iteration, all activities are clustered together into one single block, i.e., the discovered reference model.

Fig. 6.6 additionally shows cohesion values, which reflect significance of the order relations we haven chosen in different iterations. For example, in the second iteration, the cohesion we obtain when clustering activity `I` and the block containing `C` and `D`, equals 0.735. Since cohesion reflects the significance of the chosen order relation, it also expresses local fitness of the control flow in the reference model. For example, when comparing the cohesion values, it turns out to be of high significance that activity `C` precedes activity `D`, but less significant that they precede activity `I`. When reconsidering our process variants from Fig. 6.2 for example, we can make similar conclusions as the ones described here.

6.4 Mining Process Variants with Different Activity Sets

So far, our basic method for mining process variants has assumed that all variant models comprise the same set of activities. Generally, however, process variants may differ in their activity sets. In this section we discuss how to mine process variants with different activity sets. We illustrate relevant issues as well as necessary extensions of our basic method by analyzing all activities contained in process variants in Fig. 6.2.

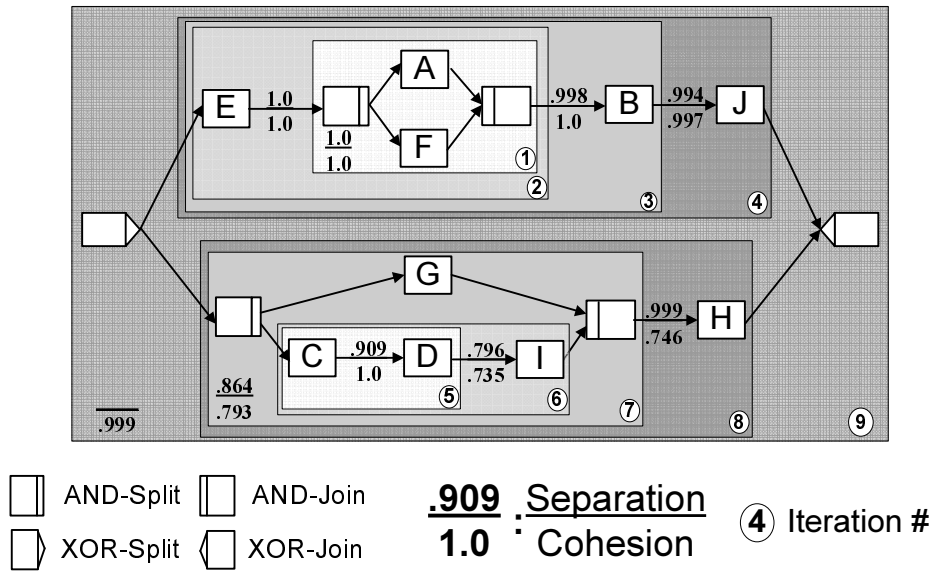


Figure 6.6: Reference Process Model Discovered Using Clustering Technique

6.4.1 Analyzing the Occurrences of Activities

One fundamental challenge is to decide which activities shall be considered in the resulting reference model and which not. Another challenge is to fix the order relations between the considered activities, which is not trivial since not all activities occur in all variant models.

For a given variants collection, we can measure how frequent each activity a_i appears using *Activity Frequency*:

Definition 12 (Activity frequency) *Let $S_i \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants. Let further $T_i = (N_i, C_i, CT_i, E_i, l_i)$ and A_i be the process structure tree and the order matrix of S_i, and w_i be the number of process instances that were executed on S_i. For each $a_j \in \bigcup_{i=1}^{n} N_i$, we define $g(a_j)$ as relative frequency with which a_j appears within the given variant collection. Formally:*

$$g(a_j) = \frac{\sum_{S_i : a_j \in N_i} w_i}{\sum_{i=1}^{n} w_i} \tag{6.7}$$

Table 6.1 shows the relative frequency of activities contained in the process variants of our running example (cf. Fig. 7.2); e.g., activity `X` is present in 80% of the variants (i.e., in S_1, S_3, S_4, S_5, and S_6), while `Z` only occurs in S_5 (i.e., 20% of the variants). Since S_4 contains a loop-block, we obtain 15% as frequency with which silent activity τ occurs (cf. Def. 8).

In MinAdept the user may set a threshold value in order to determine which activities shall be contained in the resulting reference process model and which

Activity	A	B	C	D	E	F	G	H	I	J	X	Y	Z	τ
$g(a_j)$	1	1	1	1	1	1	1	1	1	1	0.8	0.6	0.2	0.15

Table 6.1: Relative frequency of each activity within the given variant collection

not. This way we can exclude activities with low frequency if desired. For example, if we only want to consider activities with relative frequency greater than 40%, activity B as well as silent activity τ will be excluded from the reference process model (excluding τ means the loop structure will not be considered). Generally, process engineers have to set respective threshold values depending on whether they want to add more or fewer activities to the reference process model. Obviously, a good threshold value is the key to success. We discuss how to find a suitable threshold in Section 6.5.

6.4.2 Coping with Unclear Order Relations

In an aggregated order matrix, the order relations between two activities a and b are defined as 5-dimensional vector $V_{ab} = (v^0_{ab}, v^1_{ab}, v^+_{ab}, v^-_{ab}, v^L_{ab})$, and each field corresponds to the relative frequency of the respective relation type '0', '1', '+', '-' or 'L' (cf. Def. 11). When mining process variants with different activity sets, there may be pairs of activities which do not co-occur in any of the process variants and process instances respectively, but which shall be both included in the new reference process model. Since the two activities never co-occur, we are unable to derive relative frequencies for the five possible order relations; i.e., their order relations as reflected in aggregated order matrix would be (0,0,0,0,0). Regarding our example, activities Y and silent activity τ representing the loop structure do not co-occur in any process variant. Since $f(\alpha, \beta)$ (cf. Equation 6.2) will be invalid if one of the two vectors α or β equals $(0, 0, 0, 0, 0)$, we need to find ways to handle such unclear relationship:

1. **Compute separation.** When computing separations (cf. Equation 6.3), we can ignore these unclear relationships. For example, when computing $Separation(a, b)$ between activity a and b, we can see that order relation v_{ax} is an unclear one, i.e., it equals $(0, 0, 0, 0, 0)$. We therefore do not include it in the computation. To be more precise, let V be an aggregated order matrix and let $a, b \in N$ be two activities considered in V. We can define a set $\mathcal{M} = \{x \big| x \in N \setminus \{a, b\}, v_{ax} = (0, 0, 0, 0, 0) \vee v_{bx} = (0, 0, 0, 0, 0)\}$ which contains all x with v_{ax} or v_{bx} being (0,0,0,0,0). Then instead of using Equation 6.3, we use Equation 6.8 to compute $Separation(a, b)$ to ignore the influence of unclear relations.

$$Separation(a, b) = \frac{\sum_{x \in N \setminus \{a,b\}, x \notin \mathcal{M}} f^2(v_{ax}, v_{bx})}{|N| - 2 - |\mathcal{M}|} \tag{6.8}$$

2. **Compute cohesion.** When computing $Cohesion(a, b)$ (cf. Equation 6.4), we will also run into problems if their order relation is (0,0,0,0,0). In this

case we can set $cohesion(a, b)$ to 0, i.e., none of the five order relations is considered as being more significant than the others.

3. **Recompute Aggregated Order Matrix.** Since an aggregated order matrix captures the distribution of the five order relations within the collection of variants, for each vector $v_{ab}, a, b \in N$, we obtain value 1 as the sum of its elements. However, this does not apply to the unclear order relations. Therefore, when re-computing the (reduced) aggregated order matrix after the creation of a new block (cf. Section 6.3.4), we do not take such unclear order relations into account. To be more precise, let V be an aggregated order matrix and let $a, b, x \in N$ be three activities contained in V. Assume further that in one iteration of our algorithm, activities a and b are clustered into a block. Then we set new matrix element $v'_{(a,b)x} = v_{ax}$ if $v_{bx} = (0, 0, 0, 0, 0)$ holds, and $v'_{(a,b)x} = v_{bx}$ if $v_{ax} = (0, 0, 0, 0, 0)$ holds, respectively.

6.4.3 Mining Result when Considering All Activities

Fig. 6.7 shows the discovered process model S'_{all} as well as the intermediate results we obtain after every iteration of our clustering approach (indicated through the numbers at the right bottom corner of each block). Note that the discovered model contains all process activities contains in the variants. This includes the silent activity τ which represents the loop structure in S_4.

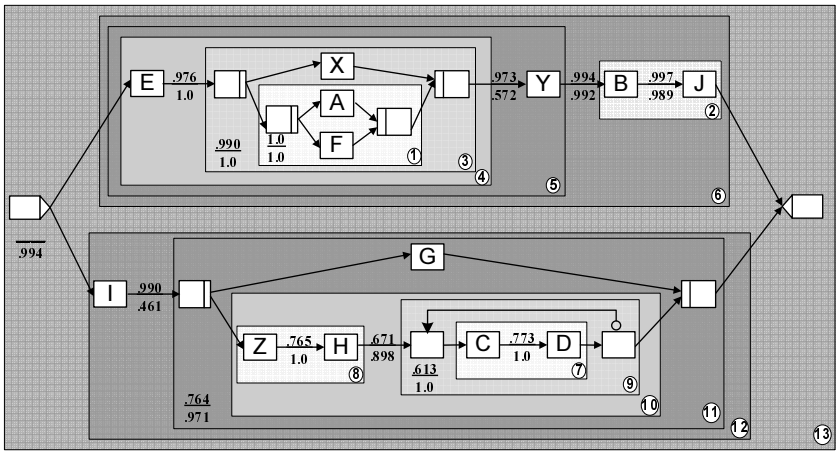


Figure 6.7: The discovered process model S'_{all} based on all activities

6.4.4 Setting Different Thresholds for Mining Reference Models

Regarding our example from Fig. 6.2, Fig. 6.8 depicts the models that can be discovered when setting different threshold values:

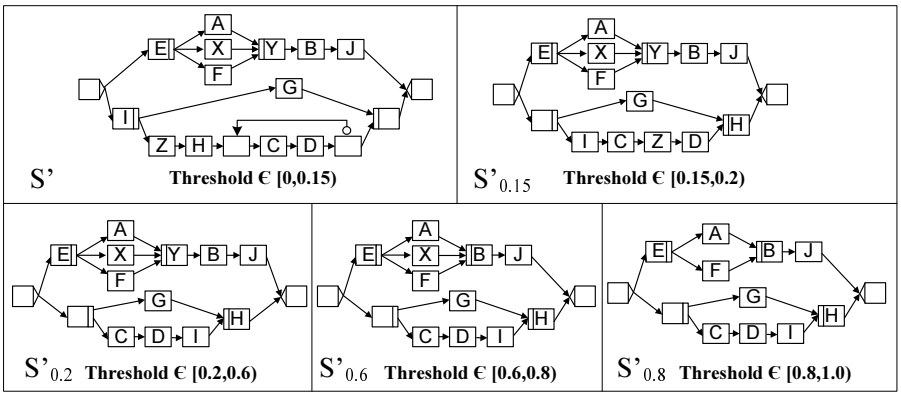


Figure 6.8: The discovered process models when setting different threshold

Regarding Fig. 6.8, process model S' contains all activities that have ever appeared in at least one of the process variants while $S'_{0.8}$ only contains activities that appear in all variants. Clearly, we discover different process models when setting different threshold values. In Section 6.5, we will discuss how to find a good threshold values in our context.

6.4.5 The MinADEPT Algorithm

Now we can formally introduce the MinADEPT algorithm for mining process variants. In pseudo code, the MinADEPT algorithm (cf. Sections 6.3 and 6.4) can be expressed as follows:

The mining starts with deciding on the set of activities to be included in the (new) reference process model. If the relative occurrence of an activity is larger than the specified threshold, we include it in the reference model. Following this we can construct the aggregated order matrix based on the order matrices of each process variant (cf. Section 6.3.1). Afterwards we apply our mining approach as described in Section 6.3, i.e., we cluster activities and blocks iteratively until all activities are contained in one block (lines 7-13 in Algorithm 1). This block then represents the reference model we have discovered. Note that when there are unclear order relations, we apply the techniques described in Section 6.4.2 to deal with them.

The complexity of the mining algorithm described in Section 6.3 corresponds to $\mathcal{O}(n^2m + n^3)$, where n equals the number of activities each variant comprises and m equals the number of variants. $\mathcal{O}(n^2m)$ corresponds to the complexity

input : A process variant collection
output: New reference process model
1 Calculate the activity occurrence for each activity captured in any of the variants;
2 **if** *Occurrence of activity* a_i > *Threshold* **then**
3 Include a_i in the reference order matrix;
4 Compute order matrix for each process variant;
5 Build the aggregated order matrix based on the selected activities;
6 **while** {*Iteration* < *number of considered activities -1*} **do**
7 Compute Separation table;
8 Determine activities to be clustered;
9 Compute cohesion between the selected activities;
10 Generate the block in the out put model;
11 Recompute the aggregated order matrix;

Algorithm 1: Process mining based on process variants

needed to build the aggregated order matrix, while $\mathcal{O}(n^3)$ corresponds to the complexity needed to mine the reference model. Consequently, our algorithm has polynomial complexity.

6.5 Evaluating Performance of the MinADEPT Algorithm through Simulation

6.5.1 Average Weighted Distance

As discussed at the beginning of this chapter, the goal of our algorithm is to discover a process model which has minimal average distance to the variants. Therefore, we first need to define **average weighted distance** between a reference process model S and its variants.

Definition 13 (Average Weighted Distance) *Let $S \in \mathcal{P}$ be a reference process model. Let further $\mathcal{M}$ be a set of process variants $S_i \in \mathcal{P}$, $i = 1, \ldots, n$, with w_i representing the number of process instances that were executed on basis of S_i. The **Average Weighted Distance** $D_{(S,\mathcal{M})}$ between S and $\mathcal{M}$ can be computed as follows:*

$$D_{(S,\mathcal{M})} = \frac{\sum_{i=1}^{n} d_{(S,S_i)} \cdot w_i}{\sum_{i=1}^{n} w_i} \tag{6.9}$$

The complexity to compute average weighted distance is $\mathcal{NP}$-*hard* since the complexity to compute the distance between two variants is $\mathcal{NP}$-*hard* (cf. Def. 9). For example, assume that we take the discovered process model S' (cf. Fig.

6.8) as new reference process model. Then distance between S' and each of the six process variants S_i (cf. Fig. 6.2) is as follows: $d_{(S',S_1)} = 6$, $d_{(S',S_2)} = 4$, $d_{(S',S_3)} = 5$, $d_{(S',S_4)} = 4$, $d_{(S',S_5)} = 7$ and $d_{(S',S_6)} = 4$. Taking variant weights into account as well (cf. Fig. 6.2), we obtain following average weighted distance being

$$(6 \times 0.2 + 4 \times 0.2 + 5 \times 0.05 + 4 \times 0.2 + 7 \times 0.15 + 4 \times 0.2) = 4.9.$$

This means we need to perform on average 4.9 high-level change operations to configure a process variant (and related instance respectively) out of the reference process model. Generally, average weighted distance between a reference model and its process variants represents how "*close*" they are.

Since computing average weighted distance has $\mathcal{NP}$-*hard* complexity (cf. Chapter 5), our algorithm does not aim at finding the global optimum, i.e., the model which has minimal average weighted distance to the variants. However, this is a rather uncritical issue. Note that most clustering techniques and other data mining algorithms aim at finding a local optimum rather than a global one since it is almost impossible to find the global optimum in reasonable time [181, 34, 110, 138]. Our suggested clustering algorithm constitutes an approach which tries to solve a complex combinatorial optimization problem in polynomial time. As benefit we can solve a large scale problem in reasonable time. However as our algorithm only searches for a local optimum, neither we can theoretically prove that the discovered model is the one with minimal average weighted distance to the variants, nor we can claim how close the discovered model is to the global optimum. In this section, we present comprehensive simulation results to show performance of our clustering algorithm in different scenarios.

6.5.2 Determining the Optimum Threshold Value

In Section 6.4, we have discussed how to mine process variants with different activity sets. One important step is to determine which activities should be considered in the reference model. In our approach, we apply a user-defined threshold to select activities: the reference model should only contain activities whose occurrence in the variants is above this threshold (cf. Algorithm 1).

Clearly, determining a good threshold is critical. If the threshold is set too low (i.e., too many non-relevant activities are considered in the reference model), this will increase efforts for configuring process variants based on the discovered model. Note that we then need to delete those none-relevant activities when configuring the variants. On the contrary, if the threshold is set too high (i.e., only few activities are considered in the reference model), configuration efforts might increase since we need to frequently insert activities to configure specific variants. Therefore, influence of the threshold value on our algorithm is of high interest.

Consider our illustrative example presented in Fig. 6.2. Fig. 6.8 shows the reference process models we can discover by setting different threshold values. When computing average weighted distance between these reference models and the six process variants, we obtain 4.75 for S', 3.75 for $S'_{0.15}$, 2.6 for $S'_{0.2}$, 2.4

for $S'_{0.6}$ and 3.0 for $S'_{0.8}$. Based on this example, we should set the threshold at around 0.6 in order to obtain a reference process model with minimal average weighted distance. Clearly, concluding this based on one example is less reliable. We perform a simulation to analyze the influence of the threshold value. In our simulation, more than 5000 process models are generated and analyzed. We describe how the simulation is setup in the next subsection, and discuss simulation results afterwards.

6.5.3 Simulation Setup

In order to obtain convincing simulation results, we consider different parameters when generating the datasets for our simulation. Amongst others, these parameters include the size and the similarity of process models. We described our data generation method in detail in Section 7.5. In summary, we generate 54 groups of datasets according to different scenarios.[7] Each dataset group contains:

1. *A reference process model*, i.e., a randomly generated model from which we configure the process variants (see Section 7.5.1 for an algorithm).
2. *100 process variants*. We generate each variant by configuring the reference model according to a particular scenario. For each group of datasets, we generate 100 process variants.

In total, we generated 5454 process models in our simulation. Note that the scenario just describes certain properties of the collection of variants configured from the reference model, but does not control the way a particular variant is generated; i.e., the 100 variants belonging to the same group are not the same, but share certain properties (e.g., having the same distance to the reference model).

Since the variants are generated by configuring a given reference model, we are able to statistically control the occurrence of activities (i.e., activity frequency) in the variants (see Section 7.5 for the method). In each group, we control the occurrence of the activities by inserting new activities with certain probabilities during the configuration of the process variants. The probabilities for inserting activities range from 0% to 100%. Consequently we obtain activities with different frequencies appearing in the variants. This way, setting different threshold values would result in different activity sets, and consequently different process model as discovered by our algorithm.

Afterwards, to each dataset group, we apply our algorithm to discover a reference model by setting threshold values to 0%, 10%, 20%, ..., and 100%. We further evaluate these results by computing the average weighted distance of the discovered model. We apply the described approach to all 54 dataset groups. In order to compare the results from the different groups, the absolute average weighted distance values are of less interest since each group has different features

[7] In Section 7.5 we have generated 72 groups of dataset based on different scenarios. However, 18 groups are not relevant in this simulation since the activity frequency are fix in these groups (groups with Parameter 3 and 4 being "Low occurrence" or "high occurrence"). Consequently, the results are summarized based on the rest 54 groups of datasets.

and covers different parts of the search space. Therefore, in each group we set the model discovered with threshold of 0% as basic model. We then evaluate the remaining models discovered using other threshold values by comparing their average weighted distances to the one of the basic model. This way, we are able to analyze all groups as a whole about how the average weighted distances of the discovered models changes according to the threshold values. In the next subsections, we present the results we obtained from the 54 groups (5400 variants) of datasets.

6.5.4 Simulation Results: Influence of Threshold Values

In this section, we analyze how the average weighted distance of the discovered reference process model changes according to the threshold values. In this context, we compare the average weighted distance of the model obtained by setting different threshold values with the one obtained by setting threshold at 0%. We have identified two types of clusters in the 54 groups of datasets with each cluster containing 27 groups of datasets.[8] The results from Fig. 6.9 are plotted as the mean of the correlative values in each group.[9]

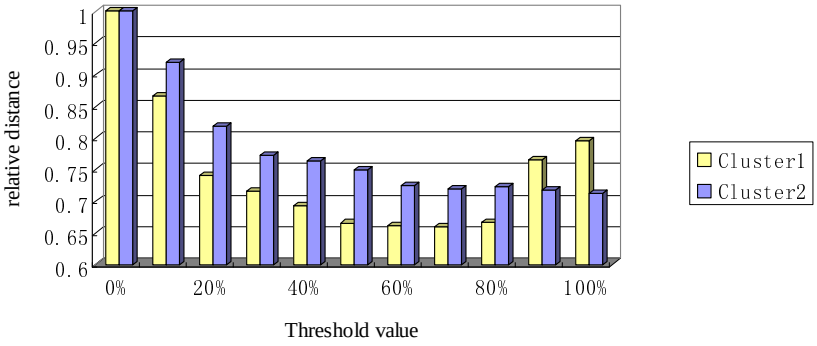


Figure 6.9: Average weighted distances of the reference models that are discovered when setting different threshold values

The relative distance of the two clusters all starts at 1 when setting the threshold to 0% (this is easy to comprehend since this is the model based on which we compare the results in each group). The relative distances in both clusters decrease with increasing threshold until around 50%. This indicates that if we filter out the activities with low occurrence the discovered reference model will have a shorter distance to the variants, and consequently will require less configuration

[8] Cluster1 contains 27 groups with parameter 3 and 4 being "High Consistency", "Positive Correlation" and "Focus on insert", where inserted activities have higher consistency; while Cluster2 contains 27 groups with Parameter 3 and 4 being "Low Consistency", "Negatively Correlation" and "Focus on 'move'", which inserted activities do not have high consistency.

[9] Data of each group is available online at http://wwwhome.cs.utwente.nl/ lic/Resources.html.

efforts. After the threshold reaches 50% the two clusters start to show different behaviors. While average weighted distance in Cluster1 begins to increase with increasing threshold, the ones in Cluster2 remains relatively stable. Such differences triggered us to perform further analysis on the datasets.

For the datasets in Cluster1, the positions where the activities are inserted in the reference model to configure the variants are relatively stable. Therefore, when considering these frequent changes in the discovered reference model, we can potentially reduce its average weighted distance to the variants. As represented in Fig. 6.9, when filtering out these activities (i.e., not considering these frequent changes (activity insertions in our case)), average weighted distance of the discovered reference increases. For the datasets in Cluster2, the positions where the activities are inserted in the reference model during process configurations are not stable. Therefore, it does not matter too much whether or not to include these activities in the discovered reference model. Since the positions of these activities are not stable, even if we include them in the new reference model, we still need to frequently move them to their respective positions in the different variants. This explains why for the datasets in Cluster2 changing the threshold value does not influence the average weighted distance too much.

Besides analyzing the means of the average weighted distances for the models resulting from each group of dataset (cf. Fig. 6.9), we have also analyzed the standard deviations of them. In both clusters the standard deviations for different threshold values are low and stable. Except the cases for which the threshold value is 0% (in this case, the standard deviation is 0, since the relative distance is always 1 in all groups), the standard deviations of the distances for all threshold values are around 0.115 (with plus or minus of 0.02). Such stable and low standard deviations indicate that in all groups the results follow almost the same trend as depicted in Fig. 6.9. Therefore, the risk for our conclusion being drawn by randomness is very low.

For a given collection of variants, knowing which cluster it belongs to can greatly help for deciding a good threshold value. However, it is very difficult to know whether the positions of an activity in a collection of variants is stable or not. We are able to obtain such information since we can control how the variants are generated in our simulation. Therefore, in order to obtain a model with shorter average weighted distance, we suggest setting the threshold to *around 50%*. In this interval, both clusters show relatively better results when compared to other values. Note that we can obtain the same result if we analyze each of the 54 groups individually.[10]

[10] As our clustering algorithm tries to solve an $\mathcal{NP}$-*hard* problem in polynomial time, another interesting question is how close the discovered model is to the real optimum. One possible solution is to enumerate all process models which can be constructed based on the given activity set, and then to evaluate the average weighted distances between the enumerated models and the variants. However, enumerating all process models can result in an enormously large collection of models (in Chapter 7, we will show that - worst case - we can obtain 2^n models by only adding one activity in a model containing n activities). Besides this, computing the average weighted distance constitutes an $\mathcal{NP}$-*hard* problem as well (cf. Def. 13). In our future work, we will design a more practical approach to evaluate the closeness of our clustering algorithm

6.5.5 Simulation Results: Running Time

Besides analyzing the influence of threshold values on the end results, we also analyze how fast our clustering algorithm runs. In our simulation, we evaluate scenarios in which the process models contain on 10-15, 20-30, and 50-75 activities in each group [11] The average running time for the 54 groups of datasets (each group contains 100 variants) is summarized in Fig. 6.10.

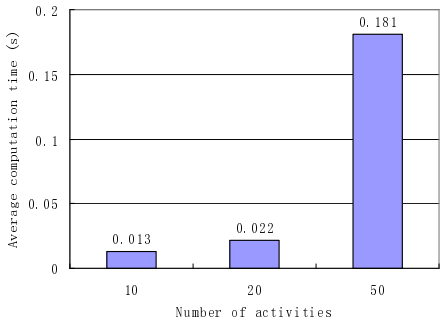


Figure 6.10: The average running time for process models with different size

We use Dell Latitude D630 laptop (2.4GHz CUP and 3.5 GB RAM) to run the simulation under Windows environment. It is clear from Fig. 6.10 that the average running time required for even large process models (containing 50-75 activities) is 0.18 seconds, which is a significant low number.

6.6 Summary

In this chapter, we have provided a cluster-based approach for mining block-structured process variants. Our overall goal is to discover a reference process model out of a collection of process variants which can be easily configured to these variants. The proposed algorithm has polynomial complexity $\mathcal{O}(n^2m+n^3)$, where n equals the number of activities each variant comprises and m equals the number of variants. This allows us to scale up when solving real-world problems.

We have further evaluated our algorithm through a simulation which comprised more than 5000 process models. Simulation results indicate that our algorithm can discover a reference model in a few seconds and we can obtain shorter average weighted distance when setting the threshold value to around 50%.

In this Chapter, we focus on evaluating the properties of our clustering algorithm. We will provide qualitative and quantitative comparisons between our clustering algorithm with other approaches in Chapter 8.

to the real optimum.

[11] According to a recent study [114], process models containing more than 50 activities bear high risk of containing errors. Following this guideline, we set the largest process model to 75 activities in our simulation.

7

Controlling the Evolution of Reference Process Models: A Heuristic Approach

7.1 Introduction

In Chapter 6 we presented a clustering algorithm which learns from past process changes by "merging" the most important parts of the process variants into one generic process model, which covers these variants best. However, when deriving a new reference process model without considering the existent one, significant structural differences between old and new reference model might occur. In many practical scenarios, too many changes of the current reference process model are not preferred due to implementation costs or social reasons. Therefore, process designers should additionally have the flexibility to control to what degree they want to change the original reference model in order to obtain a new one that better fits to the variants. In this sense, closeness of the new reference model to the old one is determined. Similarly, we determine closeness of the new reference model to the variants which act as "counterforces". Basically, this flexibility also enables designers to consider only the most relevant adaptations when evolving the reference process model.

In this chapter, we deal with our forth research question (cf. Section 1.2):

Given the original reference process model and a collection of related process variants derived from it, how can we derive a new reference process model that fits "better" to these variants? And how can we control the evolution of the reference process model, i.e., how can we enable process engineers to control to what degree the new reference model "differs" from the original one and how "close" it is to the given collection of process variants.

Fig. 7.1 describes the overall goal of this chapter. The input of our analysis solely comprises a reference process model and a collection of process variant models. We do NOT require the existence of change logs (cf. Def. 6 in Section 2.2.1) which specifically document how the reference process model has been configured into the variants. The closeness (or distance) between the reference process model and a process variant is measured in terms of the number of high-level change operations (cf. Def. 9 in Section 5.2) needed to transform the

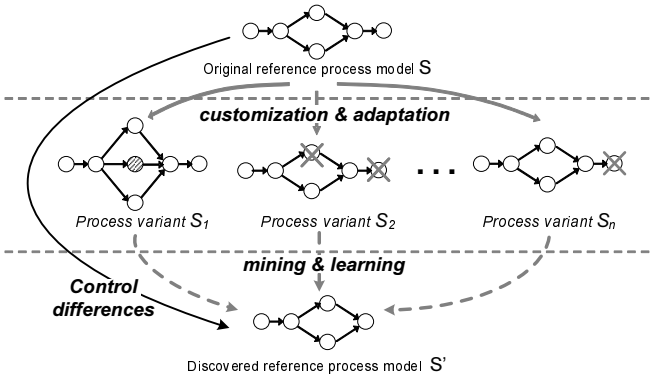


Figure 7.1: Discovering a new reference process model by learning from past process configurations and by considering the original reference model

reference process model into the respective variant. Clearly, the shorter distance is, the less efforts are needed for process adaptation.

Basically, we discover a new reference model by performing a sequence of change operations on the original one. In this context, we want to provide users with the flexibility to control to what degree the old reference model and the newly discovered one are similar, i.e., to choose how many change operations shall be applied to the old reference model to discover the new one. As benefit, we cannot only control the efforts for updating the reference process model, but also avoid Spaghetti-like model structures, which is a common challenge in the field of process mining [197, 39, 19]. Clearly, the most relevant changes, which significantly contribute to reduce the average distance between discovered reference model and variants, should be considered first and the less important ones last. Particularly, if users decide to ignore less relevant changes, the overall performance of our algorithm in respect to the described research goal shall not be influenced too much. Such flexibility to control the difference between original and discovered model constitutes a significant improvement when compared to the clustering algorithm presented in Chapter 6.

The remainder of this chapter is organized as follows. Section 7.2 introduces our heuristic search algorithm and provides a high-level overview on how it can be used for mining process variants. We describe two important aspects of our heuristics algorithm - the fitness function and the search tree - in Sections 7.3 and 7.4. To evaluate the performance of our heuristic mining algorithm, we conduct a simulation. Section 7.5 describes its setup, while Section 7.6 presents simulation results. Finally, we conclude with a summary in Section 7.7.

7.2 Overview of our Heuristic Search Algorithm

We first present an illustrating example in Section 7.2.1. Then, we give an overview of our mining algorithm in Section 7.2.3.

7.2.1 Running Example

Fig. 7.2 depicts an illustrating example. Our original reference process model S is based on common workflow patterns like AND-split, AND-join, XOR-split, XOR-join, and Loop [193]. Out of this reference model S six different process variants $S_i \in \mathcal{P}$ $(i = 1, 2, \ldots 6)$ were configured through structural adaptations. Note that these variants do not only differ in structure, but also in respect to their activity sets. For example, activity `X` appears in 5 of the 6 variants (except S_2), while activity `Z` only appears in S_5. Furthermore the six variants are weighted. In the context of our work, we define the **weight** w_i of a process variant S_i as the number of process instances executed on basis of S_i. In our example, 25 instances were executed on basis of process variant S_1, while 20 instances ran on S_2. If we only know the process variants, but have no runtime information about related instance executions, we assume variants to be equally weighted; i.e., every process variant then has weight 1. Note that structure and weight of the six process variants in Fig. 7.2 are the same as the variants in Fig. 6.2. However, the variants in Fig. 7.2 are configured from an original reference process model S, while the variants in Fig. 6.2 are not.

We can further compute the distances (cf. Def. 9) between original reference process model S and each process variant S_i. For example, when comparing reference process model S with process variant S_1 we obtain as distance five (cf. Fig. 7.2); i.e., we need to apply five high-level change operations to transform S into S_1: $delete(loop)$, $move(S, \texttt{H,I,D})$, $move(S, \texttt{I,J}, endFlow)$, $move(S, \texttt{J,B}, endFlow)$, and $insert(S, \texttt{X,E,B})$ (cf. Def. 5). As average weighted distance between S and the six variants (cf. Def. 13 in Chapter 6), we obtain 4.85. This means we need to perform on average 4.85 high-level change operations to configure a process variant (and related instance respectively) out of the reference process model.

7.2.2 Naive Approaches

Before we jump directly into our solution approach, we first discuss two naive approaches.

One naive approach is to simply pick the variant with the highest weight as the new reference process model. In our example, we would then choose S_1 (cf. Fig. 7.2). By setting S_1 as the new reference process model, as average weighted distance between S_1 and all six variants, we obtain 2.65. This value is at least better than when considering the original reference model S or any of the other five variants as new reference model. Note that the average weighted distance

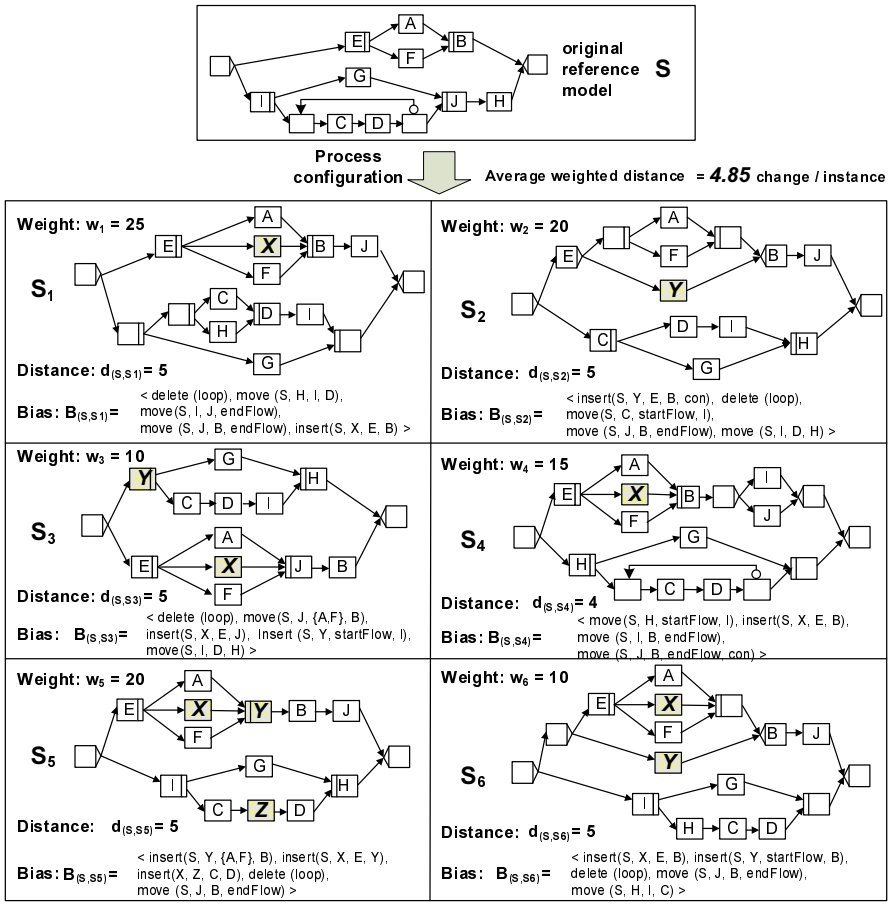


Figure 7.2: An illustrating example of a reference process model and related process variants

between S_i and the six variants are 3.8 ($i = 2$), 3.5 ($i = 3$), 3.85 ($i = 4$), 3.45 ($i = 5$) and 3.35 ($i = 6$).

In Section 7.4.3, we will compare the average weighted distance between S_1 and the model we discover using our heuristic algorithm. In addition, simply setting S_1 as the new reference process model has another disadvantage, that it becomes not possible to control the evolution of the reference process model. It is clear from Fig. 7.2 that the distance between S and S_1 is 5. However, it is not clear that which of the five change operations in the bias are more important than others. Therefore, if we are only allowed to perform, saying 3 changes, on the original reference process model, we are not able to differentiate important changes with travail ones. This problem gets even worse when considering the fact that a bias $B_{(S,S_1)}$ is only one of the possible sequences of changes to trans-

form S into S_1, and there can be a lot more other ways to realize such model transformation.

Another naive approach is to use brutal force method to enumerate all process models constructed based on the given activity set, and then to evaluate them by comparing their average weighted distances to the variants. However, enumerating all process models can result in an enormously large collection of models (in Chapter 7, we will show that - worst case - we can obtain 2^n models by only adding one activity in a model with n activities). Besides this, computing the average weighted distance is an $\mathcal{NP}$-*hard* problem as well (cf. Def. 13). This indicates that more sophisticated approach for discovering reference process models is needed.

7.2.3 Heuristic Search for Process Variant Mining

As discussed in Chapter 5, measuring the distance between two models is an $\mathcal{NP}$-*hard* problem, i.e., the time for computing the distance is exponential to the size of the process models. Consequently, the problem set out in our research question (i.e., to find a reference process model which has minimal average weighted distance to the variants) is an $\mathcal{NP}$-*hard* problem as well. When encountering real-life cases (e.g., hundreds up to thousands of variants with complex structure), finding "the optimum" would therefore be either too time-consuming or simply be not feasible. In this thesis, we present a **heuristic search algorithm** for process variant mining. Our overall goal is to find a solution which is close to "the optimum", but can be computed in a reasonable amount of time.

Heuristic algorithms are widely used in various fields of Computer Science, like artificial intelligence [110], data mining [181] and machine learning [138]. A problem employs heuristics when "it may have an exact solution, but the computational cost of finding it may be prohibitive" [110]. Although heuristic algorithms do not aim at finding the "real optimum" (i.e., it is neither possible to theoretically prove that the discovered result is the optimum nor can we say how close it is to the optimum), they are widely used in practice. Usually heuristic algorithms provide a nice balance between the goodness of the discovered solution and the computation time needed for finding it [110].

Regarding the mining of process variants, Fig. 7.3 illustrates how heuristic algorithms can be applied in our context. Here we represent each process variant S_i as single node in the two dimensional space (white node). The goal of variant mining is then to find the "center" of these nodes (bull's eye S_{nc}), which has minimal average distance to them. In addition, as discussed in Section 7.1, we also want to take the original reference model S (solid node) into account, such that we can control the difference between the newly discovered reference model and the original one. Basically, this fundamental requirement motivates us to balance two forces: one is to bring the newly discovered reference model closer to the variants (i.e., to the bull's eye S_{nc} at the right) than the old one; the other force is to "move" the discovered model not too far away from original reference model S (i.e., the solid node at left) such that it does not differ too

much from the original one. Process designers obtain the flexibility to balance these two forces, i.e., they are able to discover a model (e.g., S_c), which is closer to the variants than the old one, but which is still within a limited distance to the latter. Clearly, the change operations applied first to the (original) reference model should be more important (i.e., reduce the distance between the reference model and the variants more) than the ones positioned at the end. Consequently, if we ignore less relevant changes, we will not influence overall distance reduction between reference model and variants too much.

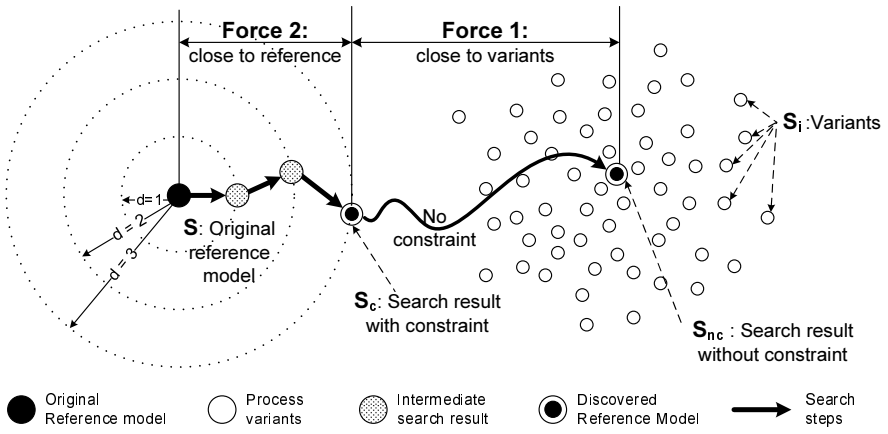


Figure 7.3: Heuristic search approach

Basically, our heuristic algorithm works as follows:

1. We use original reference model S as starting point.
2. We search for all neighboring process models with distance 1 to the currently considered reference process model. If we are able to find a better model S' among these candidate models (i.e., one which has lower average weighted distance to the given collection of process variants than S), we replace S by S'.
3. Repeat Step 2 until we either cannot find a better model or the maximally allowed distance between original and new reference process model is reached. Finally, S' corresponds to our discovered reference model S_c.

If we do not set any search limitation, our heuristic algorithm is still able to find the "center" of the variants (i.e., S_{nc}). This implies that it can be also applied to scenarios where there only exists a collection of variants, but the original reference model is not known. In this case, we can randomly select a variant S_i as starting point and search unlimitedly until we find the "center", i.e., the model with minimal average weighted distance to the given collection of variants.

Generally, most important for any heuristic search algorithm are two aspects: the *heuristic measure* and *the algorithm* that uses heuristics to search the state

space. Section 7.3 introduces our **fitness function** which measures the quality of a particular candidate model; i.e., it allows us to approximately evaluate how close such candidate model is to the given variants. Section 7.4 then introduces a **best-first search** algorithm to search the state space; i.e., how to search for a next candidate process model.

7.3 Fitness Function of our Heuristic Search Algorithm

Generally, a fitness function of any heuristic search algorithm should be quickly computable. Since search space may become very large, we must be able to make a quick decision on which path to choose next. As discussed, average weighted distance cannot be used as fitness function since complexity for computing it is $\mathcal{NP}$-*hard* (cf. Def. 13). In this section we introduce our fitness function, which can be used to approximately measure the "closeness" between a candidate reference model and the given collection of variants. In particular, it can be computed in polynomial time. Like in most heuristic search algorithms, the chosen fitness function is a "reasonable guessing" rather than a precise measurement. Therefore, in Section 7.6 we investigate correlation between fitness function and average weighted distance. In the following, we explain how to measure fitness of a candidate process model S_c.

7.3.1 Activity Coverage

Given a candidate reference process model $S_c \in \mathcal{P}$ and its process structure tree $T = (N_c, C_c, CT_c, E_c, l_c)$ we first measure to what degree its activity set N_c covers the activities that occur in the given variant collection. Note that N_c may contain silent activities if there are loops in S_c (cf. Def. 2). We denote this measure as *activity coverage* $AC(S_c)$ of S_c. Here, we first need to compute *activity frequency* (cf. Def. 12) for all activities appearing in the process variants.

Activity frequency $g(a_j)$ measures relative frequency with which a_j appears within the given variant collection. Table 7.1 shows the activity frequency of all activities contained in any of the process variants of our running example (cf. Fig. 7.2); e.g., activity `X` is present in 80% of the variants (i.e., in S_1, S_3, S_4, S_5, and S_6), while activity `Z` only occurs in 20% of the cases (i.e., in S_5). Since variant S_4 contains a loop-block, we obtain as frequency 15% with which silent activity τ occurs in all process structure trees (cf. Def. 8).

Activity	A	B	C	D	E	F	G	H	I	J	X	Y	Z	τ
$g(a_j)$	1	1	1	1	1	1	1	1	1	1	0.8	0.65	0.2	0.15

Table 7.1: Relative frequency of each activity within the given variant collection

Based on activity frequency, we can define *activity coverage* as follows:

Definition 14 (Activity coverage) *Let S_i, $i = 1, \ldots, n$ be a collection of process variants, and let $T_i = (N_i, C_i, CT_i, E_i, l_i)$ be the process structure tree of S_i. Let further $M = \bigcup_{i=1}^{n} N_i$ be the set of activities that are present in at least one of the process structure trees. Let further $T_c = (N_c, C_c, CT_c, E_c, l_c)$ be the process structure tree of candidate process model S_c. Given activity frequency $g(a_j)$, for each $a_j \in M$ the **activity coverage** $AC(S_c)$ of model S_c is defined as follows:*

$$AC(S_c) = \frac{\sum_{a_j \in N_c} g(a_j)}{\sum_{a_j \in M} g(a_j)} \tag{7.1}$$

Obviously, the value range of $AC(S_c)$ is $[0, 1]$. Let us take original reference model S as candidate model. It contains activities `A, B, C, D, E, F, G, H, I, J`, and τ (which is added when transforming S into its corresponding process structure tree (cf. Section 2.1.2)). Therefore, its activity coverage $AC(S)$, which represents to what degree it covers the activities in the variant collection corresponds to $\frac{10.15}{11.8} = 0.860$.

7.3.2 Structure Fitting

Though $AC(S_c)$ measures how representative activity set N_c of candidate model S_c is in respect to a given variant collection, it does not say anything about the structure of the candidate model (i.e., activity order relations). We therefore introduce *structure fitting* $SF(S_c)$ as second important metrics. It measures to what degree a candidate model S_c structurally fits to the given collection of variants S_i.

Based on the order matrices of process variants (cf. Def. 8), we compute again *aggregated order matrix* (cf. Def. 11) which represents a collection of process variants as a single matrix. In addition, we define the *coexistence matrix* which indicates the importance of the order relations. Finally, we describe how to measure structure fitting $SF(S_c)$ of a candidate model S_c.

7.3.2.1 Aggregated Order Matrix

For a given collection of process variants, first, we compute the order matrix of each process variant (cf. Def. 8). Regarding our running example from Fig. 7.2, we need to compute six order matrices (cf. Fig. 7.4). Due to space limitations, we only show a partial view of the order matrices here (i.e., activities `H, I, J, X, Y, Z` as well as silent activity τ representing the Loop-block). Based on the order matrices, we compute again aggregated order matrix (cf. Def. 11) which provides a distribution for the five types of order relations between every activity pairs (cf. Def. 8). Regarding our example, in 60% of all cases, `H` is a successor of `I` (as in S_2, S_3, S_5 and S_6), in 25% of all cases `H` is a predecessor of `I` (as in S_1), and in 15% of all cases `H` and `I` are contained in different branches of an XOR block (as in S_4) (cf. Fig. 7.4). Consequently, we obtain matrix element $V_{\texttt{HI}} = (0.6, 0.25, 0, 0.15, 0)$ in aggregated order matrix V (cf. Fig. 7.4).

Fig. 7.4 partially shows the aggregated order matrix V for the process variants from Fig. 7.2. Again, due to space limitations, we only consider order relations for activities `H, I, J, X, Y, Z`, and silent activity τ which represents the Loop-block. Note that since the variants in Fig. 7.2 are the same as the variants in Fig. 6.2, the two aggregated order matrices in Fig. 6.3 and in Fig. 7.4 are the same. However, we illustrate different parts of the aggregated order matrix (activities `A,B,C,F,H,I` and `J` in Fig. 6.3 and activities `H,I,J,X,Y,Z` and τ in Fig. 7.4) to better illustrate the respective algorithms.

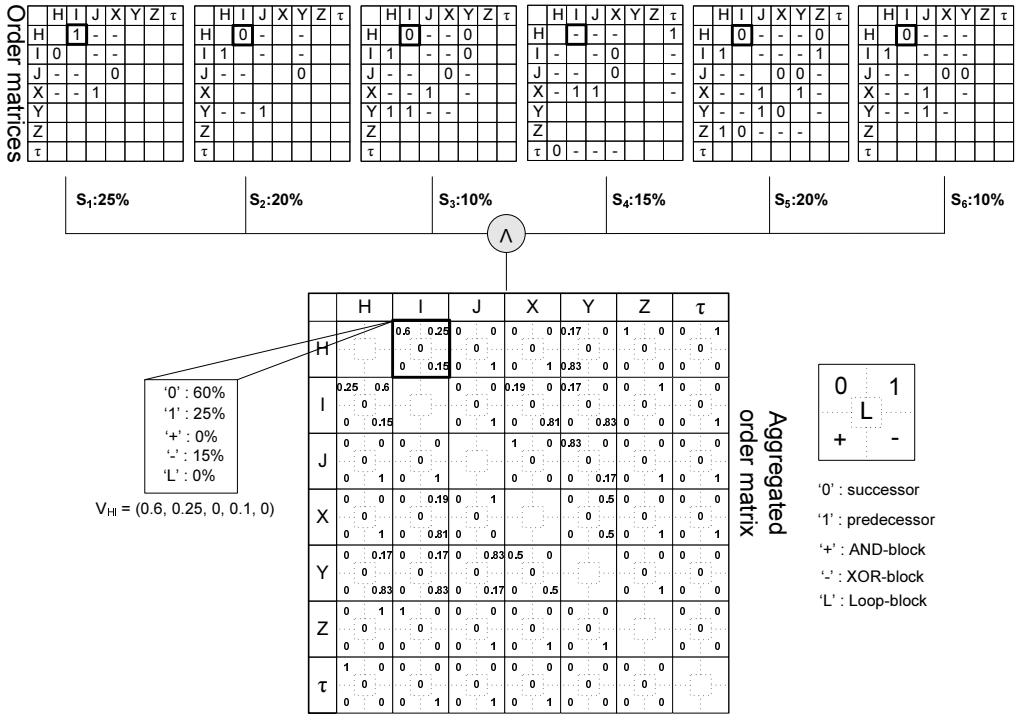


Figure 7.4: Aggregated order matrix based on process variants

7.3.2.2 Importance of the Order Relations

Generally, the order relations computed by an aggregated order matrix may be not equally important. For example, relationship $V_{\mathtt{HI}}$ between `H` and `I` (cf. Fig. 7.4) would be more important than relation $V_{\mathtt{HZ}}$, since activities `H` and `I` appear together in all six process variants while activities `H` and `Z` only show up together in variant S_5 (cf. Fig. 7.2). To cope with this, we define co-existence matrix CE in order to represent the importance of the different order relations occurring within an aggregated order matrix V.

Definition 15 (Coexistence Matrix) *Let $S_i \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants. Let further $T_i = (N_i, C_i, CT_i, E_i, l_i)$ and A_i be the process structure tree and the order matrix of S_i, and w_i be the number of process instances that were executed on S_i. The* ***Coexistence Matrix*** *of variant collection $\{S_1, \ldots, S_n\}$ is then defined as 2-dimensional matrix $CE_{m \times m}$ with $m = |\bigcup N_i|$. Each matrix element $CE_{a_j a_k}$ corresponds to the relative frequency with which activities a_j and a_k co-occur within the given variant collection. Formally: $\forall a_j, a_k \in \bigcup N_i, a_j \neq a_k$:*

$$CE_{a_j a_k} = \frac{\sum_{S_i : a_j, a_k \in N_i} w_i}{\sum_{i=1}^{n} w_i} \tag{7.2}$$

Regarding our running example, Fig. 7.5 shows the corresponding coexistence matrix. Again, we only depict the coexistence matrix for activities `H, I, J, X, Y, Z`, and silent activity τ. For instance, we obtain $CE_{\texttt{HI}} = 1$ and $CE_{\texttt{HZ}} = 0.2$. This indicates that order relation between `H` and `I` is more important than the one between `H` and `Z`.

	H	I	J	X	Y	Z	τ
H		1	1	0.8	0.6	0.2	0.15
I	1		1	0.8	0.6	0.2	0.15
J	1	1		0.8	0.6	0.2	0.15
X	0.8	0.8	0.8		0.4	0.2	0.15
Y	0.6	0.6	0.6	0.4		0.2	0
Z	0.2	0.2	0.2	0.2	0.2		0
τ	0.15	0.15	0.15	0.15	0	0	

Figure 7.5: Coexistence matrix based on process variants

7.3.2.3 Structure Fitness of a Candidate Process Models

Since we can represent a candidate process model S_c by its corresponding order matrix A_c (cf. Def. 8), we determine structure fitting $SF(S_c)$ between S_c and the variants by measuring how similar order matrix A_c and aggregated order matrix V (representing the variants) are. We can compute S_c by measuring the order relations between every pair of activities in A_c and in V.

For example, consider original reference model S as candidate process model S_c (i.e., $S_c = S$). A partial view of the corresponding order matrix A is depicted in Fig. 7.6. Obviously, $A_{\texttt{HI}}$ ='0' holds, i.e., `H` is successor of `I` in model S (cf. Fig. 7.6). Consider now aggregated order matrix V. Here the order relation between activities `H` and `I` is represented by the 5-dimensional vector

$V_{\texttt{HI}} = (0.6, 0.25, 0, 0.15, 0)$. If we now want to compare how close $A_{\texttt{HI}}$ and $V_{\texttt{HI}}$ are, we first need to build an aggregated order matrix V^c purely based on our candidate process model S_c (S in our case). Fig. 7.6 shows both the order matrix A_c and the "calculated" aggregated order matrix V^c of process model S_c (with $S_c = S$). As order relation between `H` and `I` in V^c, we obtain $V^c_{\texttt{HI}} = (1, 0, 0, 0, 0)$, i.e., `H` is always a successor of `I`. We now can compare $V_{\texttt{HI}}$ (which represents the variants) with $V^c_{\texttt{HI}}$ (which represents the reference model).

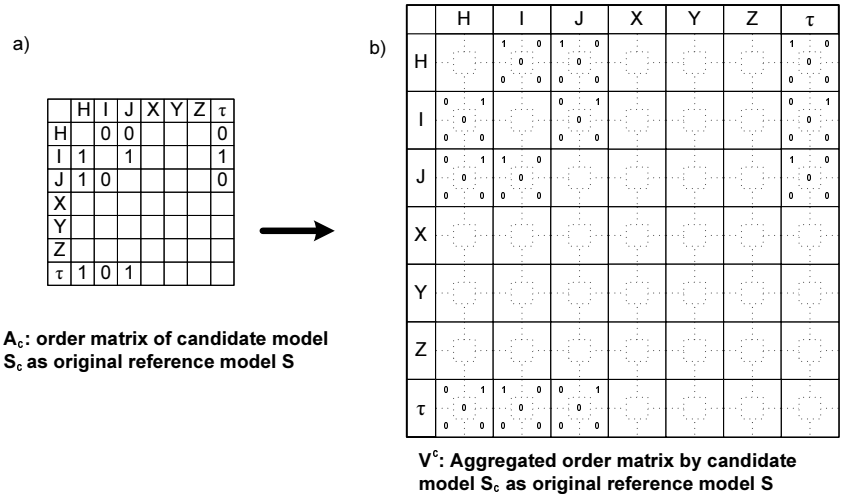


Figure 7.6: Order matrix A_c and aggregated order matrix V^c constructed by candidate model $S_c = S$

We use Euclidean metrics $f(\alpha, \beta)$ to measure the closeness between two vectors $\alpha = (x_1, x_2, ..., x_n)$ and $\beta = (y_1, y_2, ..., y_n)$ (cf. Equation 6.2). $f(\alpha, \beta) \in [0, 1]$ computes the cosine value of the angle θ between vectors α and β in Euclidean space. If $f(\alpha, \beta) = 1$ holds, α and β exactly match in their directions; $f(\alpha, \beta) = 0$ means, they do not match at all. Regarding our running example, we obtain $f(V_{\texttt{HI}}, V^c_{\texttt{HI}}) = 0.899$. This number indicates high similarity between the order relations of the candidate process model in respect to `H` and `I` and the ones captured by the variants.

Based on Euclidean metrics, which measures *similarity* between the order relations, and Coexistence matrix CE (cf. Def. 15), which measures *importance* of the order relations, we can formally define structure fitting $SF(S_c)$ of a candidate model S_c as follows:

Definition 16 (Structure Fitting) *Let $S_i \in \mathcal{P}$, $i = 1, 2, \ldots, n$ be a collection of process variants and let $T_i = (N_i, C_i, CT_i, E_i, l_i)$ be the corresponding process structure trees. Let further CE be the coexistence matrix and V be the aggregated order matrix of this variants collection. For candidate model S_c, let $T_c = (N_c, C_c, CT_c, E_c, l_c)$ be the corresponding process structure tree, and let*

$m = |N_c|$ correspond to the number of nodes in T_c. Finally let V^c be the aggregated order matrix of S_c. Then ***structure fitting*** *$SF(S_c)$ is defined as follows:*

$$SF(S_c) = \frac{\sum_{j=1}^{m} \sum_{k=1,k\neq j}^{m} (f(V_{a_j a_k}, V^c_{a_j a_k}) \cdot CE_{a_j a_k})}{m \cdot (m-1)} \quad \in [0,1] \qquad (7.3)$$

For every pair of activities $a_j, a_k \in N_c, j \neq k$, we first compute similarity of corresponding order relations (as captured by V and V_c) by means of $f(V_{a_j a_k}, V^c_{a_j a_k})$, and second the importance of these order relations by $CE_{a_j a_k}$. Structure fitting $SF(S_c) \in [0,1]$ of candidate model S_c then equals the average of the similarity multiplied by the importance of every order relation. Regarding our example from Fig. 7.2, structure fitting $SF(S)$ of the original reference model S corresponds to 0.632.

7.3.3 Fitness Function

So far, we have introduced activity coverage $AC(S_c)$ and structure fitting $SF(S_c)$ to evaluate fitness of a candidate model S_c. While $AC(S_c)$ measures to what degree the activities, occurring in the collection of variants, are covered by the candidate model S_c, $SF(S_c)$ measures to what degree S_c structurally fits to the variants, i.e., how good it covers the order relations of the different variants.

Definition 17 (Fitness) *For candidate model S_c, let $AC(S_c)$ be the activity coverage of S_c and let further $SF(S_c)$ be the structure fitting of S_c. We compute* fitness *$Fit(S_c)$ of a candidate model S_c as follows:*

$$Fit(S_c) = AC(S_c) \times SF(S_c) \qquad (7.4)$$

As $AC(S_c) \in [0,1]$ and $SF(S_c) \in [0,1]$ holds, value range of $Fit(S_c)$ is [0,1] as well. Fitness value $Fit(S_c)$ indicates how "close" a candidate model S_c is to the given collection of variants. If $Fit(S_c) = 1$ holds, candidate model S_c will perfectly fit to the variants; i.e., no additionally adaptation will be needed. Otherwise, further adaptations might be required. The higher $Fit(S_c)$ is, the closer S_c will be to the variants and the less configuration efforts will be needed. Regarding our example from Fig. 7.2, fitness value $Fit(S)$ of original reference process model S is $Fit(S) = AC(S) \times SF(S) = 0.860 \times 0.632 = 0.543$.

As fitness of a candidate model S_c is evaluated by activity coverage $AC(S_c)$ multiplied by structure fitting $SF(S_c)$, we can automatically balance the number of activities to be considered in candidate model S_c. If too many activities of low relevance (i.e., activities which only appear in a limited number of variants; e.g., activity `Z` in our example) are considered in the candidate model, we obtain a high value for $AC(S_c)$. However, in this case $SF(S_c)$ possibly decreases since coexistence values (cf. Def. 15) of less relevant activities are rather low (cf. Fig. 7.5). On the contrary, if S_c contains only few activities, $SF(S_c)$ can potentially be very high, while $AC(S_c)$ is too low in order to qualify S_c as good candidate model. Therefore, a high value for $Fit(S_c)$ does not only mean that S_c structurally fits

well to the process variants, but also that a reasonable number of activities is considered in the candidate model.

The complexity of computing $Fit(S_c)$ is polynomial: let n be the number of variants and let $m = |\bigcup_{i=1}^{n} S_i|$ be the total number of activities in the variants. Complexity to compute activity frequency (cf. Def. 12) is $\mathcal{O}(mn)$ and complexity to compute aggregated order matrix V (cf. Def. 11) is $\mathcal{O}(2m^2n)$. Based on this, complexity to compute the fitness function is $\mathcal{O}(m + 2m^2)$. Note that this is significantly lower than $\mathcal{NP}$-*hard* level complexity needed for computing average weighted distance (cf. Def. 13).

As discussed, fitness function $Fit(S_c)$ is only a "reasonable guess" rather than an exact measurement (like average weighted distance). Therefore, we analyze performance of our fitness function later in Section 7.6.

7.4 Constructing the Search Tree

We have sketched the basic steps of our heuristic mining algorithm in Section 7.2.3. In Section 7.3, we have shown how to evaluate a candidate process model S_c based on fitness function $Fit(S_c)$. In this section, we show how we can find adequate candidate process models. For that purpose we present a *best-first* algorithm which allows us to construct a search tree in such way that we can find the best candidate model in the search space.

7.4.1 The Search Tree

Let us revisit Fig. 7.3 which gives a general overview of our heuristic search approach. Starting with the current candidate model S_c and its corresponding process structure tree $T_c = (N_c, C_c, CT_c, E_c, l_c)$, in each iteration, we search for its "neighbors" (i.e., process models which have exactly distance 1 to S_c) to see whether or not we still can find a better candidate model S'_c with higher fitness value. Generally, we can construct a neighbor model for a given process model S_c by applying one insert, delete, or move operation to S_c. All activities $a_j \in \bigcup N_i$ (N_i corresponds to the activity set of process variant S_i), which have appeared in the variant collection, are candidate activities for change. Obviously, an insert operation adds an activity $a_j \notin N_c$ to S_c, while the other two operations delete or move an activity a_j already present in S_c (i.e., $a_j \in N_c$). Generally, numerous process models may result by changing one particular activity a_j on S_c. Note that the positions where we can insert ($a_j \notin N_c$) or move ($a_j \in N_c$) activity a_j can be numerous.

Section 7.4.2 provides details on how to find all process models resulting from the change of one particular activity a_j on S_c. In this section, first of all, we assume that we have already found the best process model (i.e., with highest fitness value) from all the models resulting from changing a particular activity a_j on S_c. We denote this model as the **best kid** S^j_{kid} of S_c when changing a_j (see Section 7.4.2 for computation approach of S^j_{kid}).

Our basic idea is to create all neighbor models, to evaluate each of them with the fitness function, and to finally choose the one with highest fitness value. We present a best-first algorithm to perform our heuristic variant mining (cf. Algorithm 2). To illustrate it, we use the search tree depicted in Fig. 7.7.

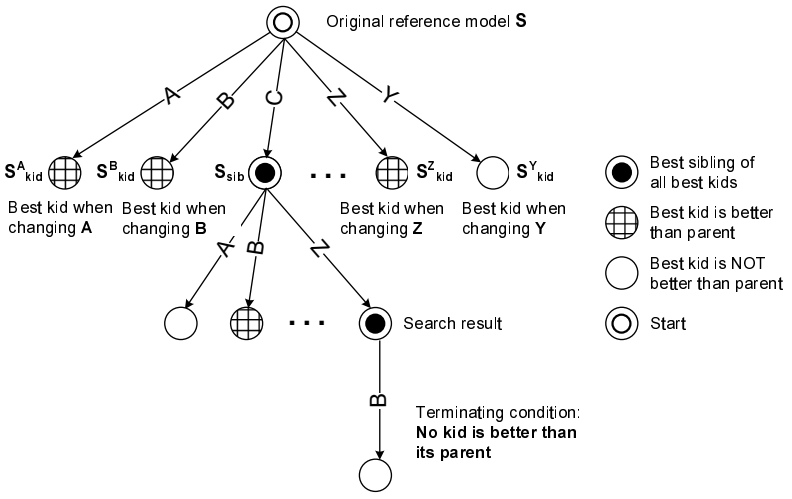


Figure 7.7: Constructing the search tree

Our search algorithm starts with setting the original reference model S as initial state, i.e., $S_c = S$ (see the node at the top of Fig. 7.7). We further define AS as active activity set, which contains all activities available for change. At the beginning, $AS = \bigcup_{i=1}^{n} N_i$ contains all activities that appear in at least one process variant S_i. For each activity $a_j \in AS$, we determine the corresponding best kid $S_{kid}^{a_j}$ of S_c when changing a_j on S_c (i.e., when deleting, moving or inserting a_j). If the best kid $S_{kid}^{a_j}$ has higher fitness value than S_c, we keep a_j in AS; otherwise, we mark it white and remove a_j from AS (cf. Fig. 7.7).[1] Afterwards, we find the best one among all the best kids $S_{kid}^{a_j}$, i.e., the one with highest fitness value. We denote this model as best sibling S_{sib} and mark corresponding activity a_s accordingly. Since model S_{sib} is the best one we can obtain by applying exactly one change operation to current candidate model S_c, we set S_{sib} as first intermediate search result and replace S_c by S_{sib} for further search (cf. Fig. 7.7, S_{sib} are marked as bull's eyes). Note that we also remove a_s from AS since this activity has now been already considered for change.

The described search method goes on iteratively, until termination condition is

[1] We can remove all nodes marked as white from active activity set AS. Consequently, we stop searching the best kids of these activities in further search steps. In principle, it is still possible that changing them later (i.e., based on another candidate model S_c') results a better model. However, chance for this is very low due to the fact that we have already enumerated all possible solutions by changing such activity on S_c. We therefore remove them from AS in order to reduce search space.

met, i.e., until we either cannot find a better model, or the allowed search distance is reached. The allowed search distance is defined by the process owner in order to control how many iterations the search method continues. Consequently, the process owners obtain the flexibility to control to what degree the discovered reference process model shall differ from the original one. The final search result S_{sib} corresponds to our discovered reference model S' (the node marked by a bull's eye and circle in Fig. 7.7).

input : A block-structured process model S; a collection of block-structured process variants S_i and their corresponding process structure trees $T_i = (N_i, C_i, CT_i, E_i, l_i)$, $i = 1, \ldots, n$; allowed search distance d ;
output: Resulting process model S'

1 $AS = \bigcup_{i=1}^{n} N_i$ /* Define AS as active activity set */;
2 $S_c = S$ /* Define initial candidate model */;
3 $t = 1$ /* Define initial search step */ ;
4 **while** $|AS| > 0$ *and* $t \leq d$ **do** /* Search condition */;
5 $S_{sib} = S_c$ /* Set S_c as initial S_{sib} */ ;
6 Define a_s as the selected activity ;
7 **foreach** $a_j \in AS$ **do**
8 S_{kid} = FindBestKid(S_c) ;
9 **if** Fitness(S_{kid}) > Fitness(S_c) **then**
10 **if** Fitness(S_{kid}) > Fitness(S_{sib}) **then**
11 $S_{sib} = S_{kid}$;
12 $a_s = a_j$;
13 **else**
14 $AS = AS \setminus \{a_j\}$;
/* Best kid not better than its parent */
15 **if** Fitness(S_{sib}) > Fitness(S_c) **then**
16 $S_c = S_{sib}$; /* Initiate next iteration */ ;
17 $AS = AS \setminus \{a_s\}$;
18 **else**
19 break ;
20 t = t+1 ;

Algorithm 2: Heuristic search algorithm for variant mining

7.4.2 Options for Changing one Particular Activity

Section 7.4.1 has shown how to construct a search tree by comparing the best kids $S_{kid}^{a_j}$. This section discusses how to find such best kid $S_{kid}^{a_j}$ when changing a particular activity a_j, i.e., we discuss how to find the "neighbors" of a candidate

model S_c by performing one high-level change operation (cf. Def. 5) on a_j. The best kid $S_{kid}^{a_j}$ is consequently the one with highest fitness value among all considered models.

Regarding a particular activity a_j, we consider three types of basic change operations: *delete, move* and *insert* activity (cf. Section 7.4.1). The neighbor model resulting through deletion of an activity $a_j \in N_c$ can be easily determined by removing a_j from the process model and the corresponding order matrix; furthermore, movement of an activity can be simulated by its deletion and subsequent re-insertion at the desired position. Thus, the basic challenge in finding neighbors of a candidate model is to apply one activity *insertion* such that *block structuring* and *soundness* of the resulting model can be maintained. Obviously, for a particular activity a_j, the positions where we can (correctly) insert it into candidate model S_c are the subjects of our interest. Inserting a_j at a (correct) position within S_c results in one neighbor model. Therefore, finding all neighbors first requires finding all valid positions where we can correctly insert a_j in S_c.

Fig. 7.8 provides one example. Given a process model S, we would like to find all process models that may result when inserting activity `X` into S. We apply the following two steps to "simulate" insertion of an activity:

1. First, we enumerate all possible blocks the candidate model S contains. A block can be an atomic activity, a self-contained part of the process model, or the process model itself (Algorithm 3 enables enumeration of all possible blocks of a process model). Note that the number of possible candidate blocks can become very large; e.g., hundreds of potential blocks may exist for a process model containing 50 activities.
2. After having determined all blocks of the current model we can simulate all possible insertions of activity `X`. For this purpose, we can cluster `X` with each block and position it in relation to this block, i.e., we can set order relation between `X` and selected block B to $\diamond \in \{0, 1, +, -\}$, or assign 'L' to $\diamond$ if `X` is a silent activity τ representing a loop-block (cf. Def. 8). This way, we obtain one neighbor model S' by inserting `X` to the respective position in S such that it forms another block together with B.

Following these two steps, we can guarantee that the resulting process model is sound and block-structured. Every time we cluster an activity with a block, we actually add this activity to the position where it can form a bigger block together with the selected one, i.e., we replace a self-contained block of a process model by a bigger one. Consider our example from Fig. 7.8a. Among the determined blocks, we can find the sequential block defined by activities `C` and `D` (Step 1). Then we can cluster activity `X` with this block using order relation $\diamond =$ "0", for example (Step 2). Consequently, we obtain S' as one neighbor of S (cf. Fig. 7.8). Note that for every block we enumerated in Step 1, we can cluster activity `X` with it by one of the order relations $\diamond$ to obtain a neighbor model S'. Therefore, if we are able to find all blocks process model S contains, we can find all neighbor models by inserting `X` into process model S. In the following, we describe these two steps in detail.

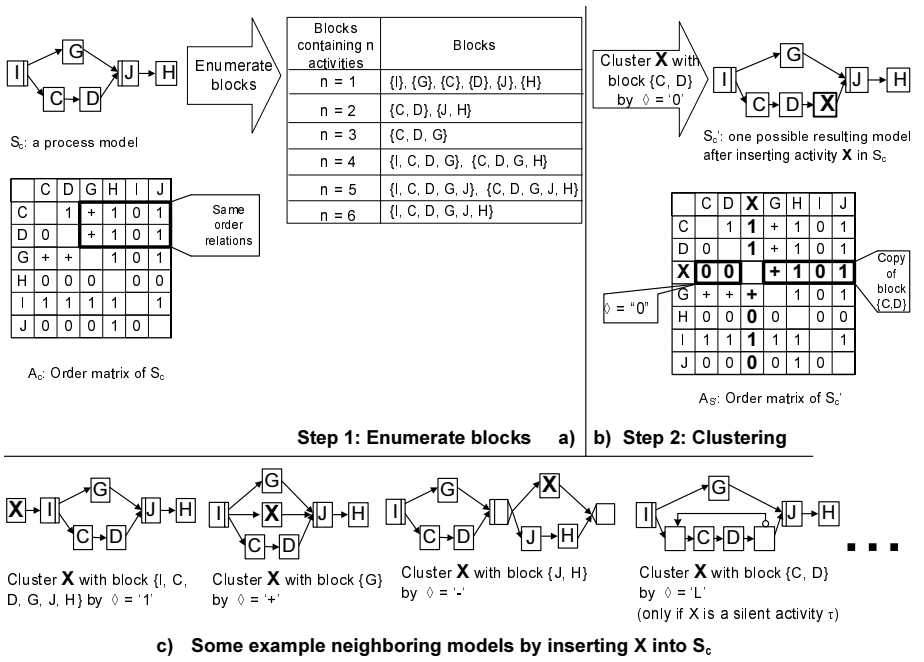


Figure 7.8: Finding the neighboring models by inserting `X` into process model S

7.4.2.1 Step 1: Block-enumerating Algorithm

We now present an algorithm to enumerate all possible blocks of a process model S. Let $S \in \mathcal{P}$ be a block-structured process model and $T = (N, C, CT, E, l)$ be its corresponding process structure tree. $N = \{a_1, \ldots, a_n\}$. Let further A be the order matrix of S. Two activities a_i and a_j can form a block if and only if the following holds:

$$\forall a_k \in N \setminus \{a_i, a_j\} : A_{a_i a_k} = A_{a_j a_k} \tag{7.5}$$

i.e., two activities can form a block if and only if they have exactly same order relations to remaining activities. Consider our example from Fig. 7.8a. Here activities `C` and `D` can form a block, since they have same order relations to remaining activities `G, H, I`, and `J`.

The block-enumerating algorithm is depicted in Algorithm 3. Let us first define BS_x as the set containing all blocks comprising exactly x activities. In its initial state, each activity forms a single block by its own (Line 2) and consequently we obtain BS_1 (Line 3). The algorithm starts by computing BS_2 (blocks containing 2 activities) and continues iteratively to compute BS_i until it reaches its upper boundary $i = n$. In each iteration, we can determine a block containing x activities by merging two disjoint blocks containing j and k activities respectively ($i = j + k$) (Line 7). For example, a block containing 2 activities can only be obtained by merging two blocks of which each contains 1 activity. Or

input : A block-structured process model S, its corresponding process structure tree $T = (N, C, CT, E, l)$ and its order matrix A
output: A set BS with all possible blocks

```
1  Define BS_x be a set of blocks containing blocks with x activities.
   x = (1, ..., n);
2  Define each activity a_i as a block B, i = (1, ..., n) ;
3  BS_1 = {B_1, ..., B_n}.                          /* initial state */ ;
4  for i = 2 to n do                                  /* Compute BS_i */
5      let j = 1; let k = i;
6      while j ≤ k do
7          k = i - j /* A block containing k activities can only be
           obtained by merging blocks containing i and j
           activities */;
8          foreach (B_j, B_k) ∈ BS_j × BS_k do /* judge whether B_j and
           B_k can form a block */;
9              merge = TRUE;
10             if B_j ⋂ B_k = ∅ then                    /* Disjoint?  */
11                 foreach (a_α, a_β, a_γ) ∈ B_j × B_k × (N \ B_j ⋃ B_k) do
12                     if A_{a_α a_γ} ≠ A_{a_β a_γ} then
13                         merge = FALSE /* two blocks con merge only
                           if they show same order relations to the
                           activities out side the two blocks */;
14                         break ;
15             else
16                 merge = FALSE;
17             if merge = TRUE then
18                 B_p = B_j ⋃ B_k;
19                 BS_i = BS_i ⋃ B_p;
20         j = j + 1 ;
21 BS = ⋃_{x=1}^{n} BS_x
```

Algorithm 3: Block enumerating algorithm

we can only obtain a block containing 5 activities by merging two disjoint blocks containing either 1 and 4 activities respectively or 2 and 3 activities respectively (Lines 5 - 20). Lines 9 to 19 check whether or not two blocks B_j and B_k can be merged. This is possible iff any activities $a_\alpha \in B_j$ and $a_\beta \in B_k$ show same order relations to the remaining activities outside the two blocks. Otherwise (Line 16), B_j and B_k cannot form a block (i.e., merge = $FALSE$). Until we obtain all sets of blocks BS_x with $x = 1, \ldots, n$ activities per block, we can define set BS as $BS = \bigcup_{x=1}^{n} BS_x$. Consequently BS corresponds to all blocks, model S contains

(Line 21).[2]Consider the example from Fig. 7.8a. For model S, all possible blocks are enumerated. As activities C and D show same order relations in respect to remaining activities in order matrix A_s, for example, they may form a block. Or, blocks {C, D} and {G} show same order relations in respect to remaining activities H, I and J; therefore they can form a bigger block {C, D, J}. As S contains 6 activities, its blocks are organized in 6 groups with blocks of different sizes.

7.4.2.2 Step 2: Cluster Inserted Activity with a Block

In Step 1, we have shown how to enumerate all possible blocks for a given candidate model S_c. Based on this, we describe where we can insert a particular activity a_j in S_c such that we obtain a sound and block-structured model again.

Assume that we want to insert activity X in S (cf. Fig. 7.8). To ensure block structure of the resulting model, we "cluster" X with an enumerated block, i.e., we replace one of the previously determined blocks B by a bigger block B' containing both B and X. In the context of this clustering, we set order relation between B and X to $\diamond \in \{0, 1, +, -\}$ (see Def. 8), i.e., the order relations between X and all activities of B are defined by $\diamond$. One example is given in Fig. 7.8b, where inserted activity X is clustered with block {C, D} by order relation $\diamond$ = "0", i.e., we set X as successor of the sequence block containing activities C and D. To realize this clustering, we have to set order relations between X on the one hand and activities C and D from the selected block on the other hand to "0". Furthermore, order relations between X and remaining activities are same as for C and D respectively. Afterwards these three activities form a new block {C, D, X} replacing the old one (i.e., {C, D}). This way, after inserting X into S, we obtain a sound and block-structured process model S'.

Fig. 7.8b shows one resulting model S' which we obtain when inserting X in S. Obviously, S' is not the only neighboring model here since we can insert X at different positions in S; i.e., for each block S enumerated in Step 1, we can cluster it with X by any one of the four order relations $\diamond \in \{0, 1, +, -\}$, or 'L' if X is a silent activity τ which represents Loop-block. Regarding our example from Fig. 7.8, S contains 14 blocks. Consequently, the number of models that may result when inserting X in S equals $14 \times 4 = 56$, or $14 \times 1 = 14$ if X is a silent activity τ which represents Loop-block; i.e., we obtain 56 potential models (or 14 if X is a silent activity) by inserting X into S [3] . Fig. 7.8c shows some neighboring models of S. Note that the resulting models are not necessarily unique, i.e., it is

[2]Worst-case, the complexity of this algorithm is 2^n where n corresponds to the number of activities. However, this worst-case scenario will only occur if any combination of activities may form a block (like a process model for which all activities are ordered in parallel to each other). During our simulation, in most cases we were able to enumerate all blocks of a process model within few milliseconds. This indicates that complexity is low in practice.

[3]During the implementation of this algorithm, we also remove all blocks which consist only one silent activity τ. Because a silent activity τ exists only to represents the Loop-block in a process model, it is therefore not possible to cluster an activity only with this silent activity to form a block.

possible that some of them are the same. However, this is not an important issue in our context since fitness function $Fit(S_c)$ can be quickly computed. Therefore, some redundant information does not significantly decrease performance of our heuristic search algorithm.

7.4.3 Search Result for our Running Example

Regarding our example from Fig. 7.2, we now present the search result we obtain when applying our heuristics search algorithm. Fig. 7.9 does not only show the finally resulting model, but also depicts all intermediate process models discovered during the search. Note that in this scenario, we do not set any limitation on the number of search steps, i.e., we allow the algorithm to go as far as possible to find the best reference model.

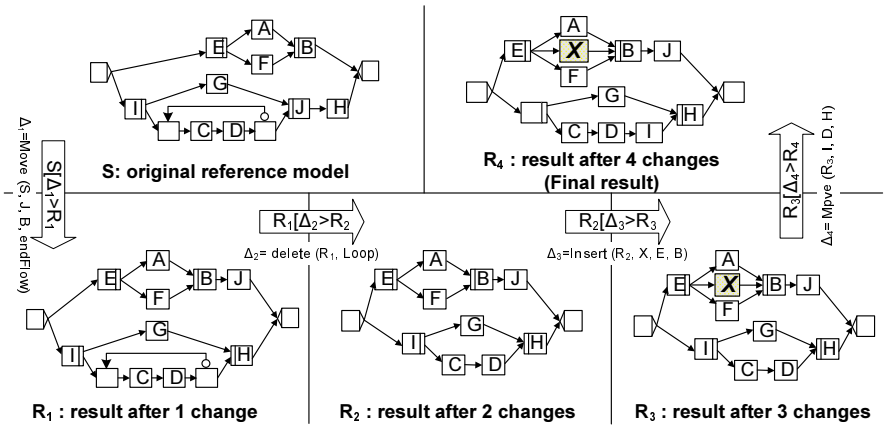


Figure 7.9: Search result by every change operations

Fig. 7.9 shows the evolution of the original reference model S. The first operation $\Delta_1 = move(S, \texttt{J,B}, endFlow)$ changes S into intermediate result model R_1. According to Algorithm 2, R_1 constitutes that neighbor model of S which can be derived by applying one valid change operation to S and which shows highest fitness value in comparison to all other neighbor models of S. Using R_1 as next input for our algorithm, we discover process model R_2. Here, change operation $\Delta_2 = delete(R_1, Loop)$ is applied. Based on R_1, the search algorithm discovers R_3 with $\Delta_3 = insert(R_2, \texttt{X, E, B})$. Finally, we obtain R_4 by performing change $\Delta_3 = move(R_2, \texttt{I,D,H})$ on model R_3. Since we cannot find a "better" process model by changing R_4 anymore, we obtain R_4 as final result. Note that if we set constraints on allowed search steps (i.e., we only allow to change original reference model by maximal d change operations), the final search result would be as follows: R_d *if* $d \leq 4$ *or* R_4 *if* $d > 4$. We further compare the original reference model S and all (intermediate) search results in Table 7.2.

We first show the fitness value of all the models in Fig. 7.9. As our heuristic search algorithm is based on finding process models with better fitness values, we can observe improvements of the fitness values with each search step. The fitness value $Fit(S)$ increases from 0.543 (model S) to 0.687 (model R_1), and then to 0.805 (model R_2) and to 0.844 (model R_3). Finally, it reaches 0.859 (model R_4). Though such fitness value is only a "reasonable guessing" of how good the result model is, the improvement of the fitness value at least indicates that discovered models is assumed to get better in each iteration.

Still, we need to examine whether or not the discovered process models are indeed getting better. We therefore compute the average weighted distance between the discovered model and the variants, which is a precise measurement in our context. From Table 7.2, the improvement of average weighted distances after applying the above changes becomes clear, i.e., the average weighted distance drops monotonically from 4.85 (when considering model S) to 2.4 (when considering model R_4). Measuring the average weighted distance shows that for the given example, the algorithm performs as expected. Note that R_4 also has shorter average weighted distance than S_1 which is the variant with the highest weight value.

One important reason to design a heuristic search algorithm in our context was to be able to only consider the most relevant change operations, i.e., the important changes (reducing average weighted distance between reference model and variants most) should be discovered at beginning while the trivial ones should be either ignored or be put at the end (cf. Section 7.1). We therefore additionally evaluate *delta-fitness* and *delta-distance*, which indicate the relative improvement of fitness values and the reduction of average weighted distance for every iteration of the algorithm. For example, the first change operation Δ_1 changes S into R_1, and consequently improves fitness value (delta-fitness) by 0.143 and reduces average weighted distance (delta-distance) by 0.9. Similarly, Δ_2 reduces average weighted distance by 0.7, Δ_3 by 0.6 and Δ_4 by 0.25. It is obvious that the delta-distance is monotonically decreasing as the number of change operations increases. This indicates that the important changes are performed at beginning of the search, while the less important ones are performed at the end.

Another important feature of our heuristic search is its ability to automatically decide on which activities shall be included in the reference model. A predefined threshold or filtering of the less relevant activities in the activity set

	S	R_1	R_2	R_3	R_4
Fitness	0.543	0.687	0.805	0.844	0.859
Average weighted distance	4.85	3.95	3.25	2.65	2.4
Change Operation		Move	Delete	Insert	Move
Delta-fitness		0.143	0.118	0.039	0.009
Delta-Distance		0.9	0.7	0.6	0.25

Table 7.2: Search result by every change

are not needed. In our example, `X` is automatically inserted, while the Loop is automatically deleted. The only concern in our heuristic variant mining is to reduce the average weighted distance, i.e., the three change operations (insert, move, delete) are automatically balanced based on their influence on the reduction of average weighted distance. This is also a significant improvement when compared to many other process mining techniques in which preprocessing of trivial activities should be conducted before performing the mining [98, 195].

7.5 Simulation Setup

Clearly, using only one example to measure performance of our heuristic mining algorithm is far from being enough. Since computing average weighted distance is at $\mathcal{NP}$-*hard* level, the fitness function, whose calculation needs polynomial time, can be only an approximation for it. Generally, we must not assume that improvement of the fitness value always results in reduced average weighted distance. Therefore, we first have to analyze *to what degree fitness improvement (delta-fitness) correlates with reduction of average weighted distance (delta-distance)?*

We further want to analyze whether our algorithm scales up. Clearly, search space is significantly larger and thus it takes longer to find the result if we have to cope with a large number of variants with dozens up to hundreds of activities. Therefore, it is important to analyze is whether performance of our algorithm decreases when facing larger models, i.e., we check *whether correlation between delta-fitness and delta-distance depends on the model sizes?*

Furthermore, we need to evaluate whether or not most important changes (i.e., the change operations which largely reduce average weighted distance) are performed at the beginning of our search. If this is the case, running our algorithm will still provide good results when setting search limitations or filtering out the change operations performed at its end. Therefore, our third research question is as follows: *to what degree are most relevant change operations positioned at the beginning of the search steps?*

Finally, we investigate whether we can further improve performance of our algorithm by applying other data mining or artificial intelligence techniques, i.e., we try to adopt the concept of "pruning" as commonly used in data mining and artificial intelligence [181, 110]. In our context, we can "prune" the cases in which delta-fitness is not nicely correlated with delta-distance, and consequently improve performance of our algorithm by adapting it to such cases. Therefore, our last research question is: *How can we improve performance of our heuristic mining algorithm by adopting the concept of "pruning"?*

We try to answer these four questions using *simulation*. In a simulation, "we numerically excise the model for the inputs and see how they affect the outputs" [95]. Simulation is often applied in system design, analysis and evaluation, and is one of the most widely used, if not *the* most widely used, techniques in operations research and management science [95]. In the context of our research, we can provide statistical answers for our four research questions by generating thousands

of process models as input for our analysis.

Section 7.5 describes the setup of our simulation. Simulation results, are presented in Section 7.6. In general, we create 72 dataset groups based on different scenarios. Each of these groups consists of 1 reference model and 100 variants configured out of it. In total, we consider 7272 process models in our analysis.

7.5.1 Generating Reference Process Models

Our general idea of randomly generating block-structured reference models is to cluster blocks, i.e., we randomly cluster activities and blocks respectively into a bigger block. This clustering continues iteratively until all activities (blocks) are clustered (see Algorithm 4 for details).

input : Set of activities a_i the process model to be generated should contain, $i = (i, \ldots, n)$
output: Sound and block-structured process model S

1 Define each activity a_i as a basic block B_i, $i = (1, \ldots, n)$;
2 Define set $\mathcal{B} := \{B_1, \ldots, B_n\}$ `/* initial state */` ;
3 **while** $|\mathcal{B}| > 1$ **do**
4 randomly selected two blocks $B_i, B_j \in \mathcal{B}$;
5 randomly select an order relation $\diamond \in \{0, 1, +, -\}$, or $\diamond \in \{L\}$ if either B_i or B_j is a block which consists only a silent activity τ to represent Loop (cf. Def. 8) ;
6 build block B_k which contains sub-blocks B_i and B_j having order relation $\diamond$;
7 $\mathcal{B} := \mathcal{B} \setminus \{B_i, B_j\}$;
8 $\mathcal{B} := \mathcal{B} \bigcup \{B_k\}$;
9 $S := B_0$ with $B_0 \in \mathcal{B}$

Algorithm 4: Randomly generating a reference model

To illustrate how Algorithm 4 works, we show an example in Fig. 7.10. As input, a set of activities {`A,B,C,D,E`} is given. The goal is now to randomly construct a block-structured process model S out of this activity set. The algorithm starts with considering each activity a_i as basic block B_i and adding these basic blocks to set $\mathcal{B}$ (Lines 1-2): $\mathcal{B} = \{\{\texttt{A}\}, \{\texttt{B}\}, \{\texttt{C}\}, \{\texttt{D}\}, \{\texttt{E}\}\}$. Following this, Algorithm 4 randomly selects two blocks B_i and B_j (Line 4) and clusters them using a randomly chosen order relation $\diamond$ (Lines 5-6). Regarding our example, blocks {`B`} and {`C`} are first selected to construct new block {`B, C`} with randomly chosen order relation '1' (i.e., `B` precedes `C`). The newly created block {`B, C`} then replaces blocks {`B`} and {`C`} within set $\mathcal{B}$, i.e., we obtain $\mathcal{B} = \{\{\texttt{A}\}, \{\texttt{B}, \texttt{C}\}, \{\texttt{D}\}, \{\texttt{E}\}\}$ (Lines 7-8). This procedure (Lines 4-8) is repeated until block set $\mathcal{B}$ only contains one single block $B_0 = \{\texttt{A,B,C,D,E}\}$. B_0 then represents our randomly generated process model S (Line 10). Fig. 7.10 shows this model as well as the blocks constructed in each iteration.

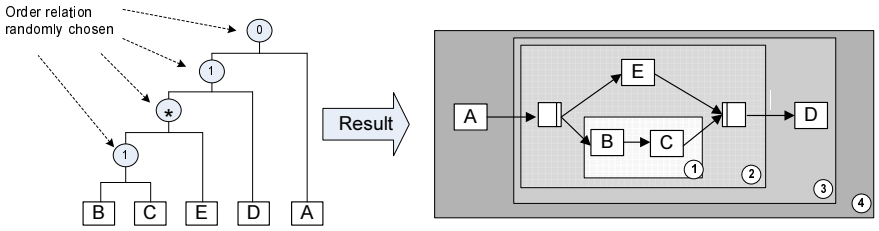


Figure 7.10: Example of generating a random process model

In practice, certain order relations are used more often than others; e.g., the predecessor/successor relations occur more frequently than AND- or XOR-splits [236]. When randomly generating process models, we take this into account as well. Rather than assuming a uniform distribution of the different order relations for the clustering of the two blocks, we set the probability for choosing an AND-split ($\diamond =' +'$) and the one for choosing an XOR-split ($\diamond =' -'$) respectively to 10%, while predecessor-successor relationships ($\diamond = \{0, 1\}$) are chosen with probability of 80%.

7.5.2 Parameters for Generating Process Variants

Taking a randomly generated reference process model, our simulation controls the way variants are configured. This can be done by adjusting a number of parameters; e.g., on how many change operations shall be performed when configuring a particular variant or the position within the model where activities shall be moved or inserted at. Altogether, we consider four parameters:

1. **Parameter 1 (Size of Process Models)**. The size of a process model variant (i.e., the number of its activities) might influence performance. Therefore, we need to check behavior of our algorithm when applying it to variants of different size. This is particularly important to evaluate scalability of our algorithm, i.e., we need to check whether correlation of delta-fitness and delta-distance depends on model size.

2. **Parameter 2 (Similarity of Process Variants)**. This parameter controls how "close" the variants are; e.g., whether or not they are similar to each other. In this context, *similarity* measures change distance, i.e., how difficult it is to configure one variant into another (cf. Def. 9).

3. **Parameter 3 (Activity Occurrence)** This parameter controls the probability of each activity for being involved in changes. If an activity frequently changes when configuring the variants, this should be considered in the discovery of the new reference model.

4. **Parameter 4 (Activity Consistence)** This parameter controls "homogeneity" of the applied insert and move operations; e.g., whether an activity

is moved to (or inserted at) same or similar positions. Parameter 4 helps us to examine whether such homogeneity influences search results.

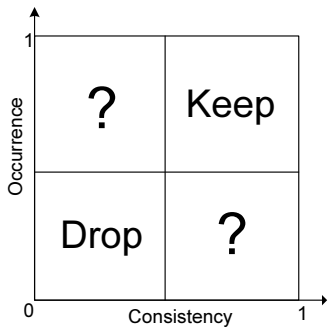


Figure 7.11: Consistency and occurrence

As Parameters 1 and 2 are rather intuitive, Parameters 3 and 4 are more difficult to comprehend. Consider Fig. 7.11. Assume that activity a_j is frequently inserted when configuring variants (high occurrence), and that its insertion position is rather constant (high consistency). Then our heuristic algorithm should also insert a_j into the reference model with high probability (upper-right quadrant of Fig 7.11). On the contrary, if an activity appears in only few variants (low occurrence) and its positions within those variants are varying a lot (low consistency), we should ignore it (lower-left quadrant of Fig. 7.11). The unclear cases concern the other two quadrants in the value space (marked with question marks in Fig. 7.11). Generally, it is difficult to decide whether or not we should insert an activity with high frequency, but very instable positions. Even if we add this activity to the reference model, we need to move it quite frequently due to its instable position; i.e., adding such activity does not necessarily reduce average weighted distance. Furthermore, if an activity does not appear frequently within the given variant collection, but its position within the variant is constant, it is also difficult to determine whether we should add this activity to the reference model. If we insert it in such case, we need to delete it rather often during variant configuration. In the following subsections, we explain how we simulate the different scenarios to cover the complete value space.

7.5.3 Parameter Settings

When configuring a variant out of a randomly created reference model, we vary values of the four parameters described in Section 7.5.2.

Parameter 1 (Size of Process Models) This parameter controls how many activities shall be contained in the original reference model and thus controls size of process variants. We consider three options:

- **Small-sized** reference models (10 activities)

- **Medium-sized** reference models (20 activities)
- **Large-sized** reference models (50 activities)

Our simulation uses same reference model for scenarios containing process models of same size. This way, we want to avoid the influence of the randomly generated reference model. According to [114], process models containing more than 50 activities show high risk of errors; i.e., it is not recommended to design such large models. Following this guideline, we set the largest reference model to 50 activities in our simulation. Still variants can have different activity sets in comparison to the reference model since their configuration also employs insert and delete operations.

Parameter 2 (Similarity of Process Variants) Closeness between variants is measured by the total number of change operations we apply when generating the variants (cf. Def. 9). Three possible choices exist:

- **Small-change** (10% changed activities)
- **Medium-change** (20% changed activities)
- **Large-change** (30% changed activities)

For datasets comprising large-sized process variants with 50 activities, medium-change means that we need to apply 10 change operations to the reference model in order to configure a particular process variant. This way, we can control distance between reference model and variants. Indirectly, we can further control similarity between the variants.

Our simulation allows to change up to 30% of the activities when configuring a particular variant. From this, we can conclude that the difference between two variants can be up to 60%. We assume that in practice 60% difference between variants constitutes a significantly large number. Note that process variants usually have common parts, i.e., they are more or less similar to each other. [56] shows similar results concerning the degree of deviations between reference model and variants. Here, it is a headquarter policy that 80% of the processes need to comply with the global process, while 20% deviations are tolerated to adjust the process to local regulations.

Parameter 3 (Activity Occurrence) and Parameter 4 (Activity Consistency) Fig. 7.11 depicts the influence *activity occurrence* and *activity consistency* have on our algorithm. In order to analyze their relationship, we have designed eight different scenarios to cover the space as illustrated in Fig. 7.11. The scenarios are depicted in Fig. 7.12.

Fig. 7.12a shows four simple scenarios. Each of them is constructed by keeping either occurrence or consistency stable while changing the other dimension:

1. **Low Occurrence:** Occurrence of activities is kept at 30% while their consistency varies from 0 to 80%.
2. **High Occurrence:** Occurrence of activities is kept at 70% while their consistency varies from 0 to 80%.
3. **Low Consistency:** Consistency of activities is kept at 30% while their occurrence varies from 0 to 80%.

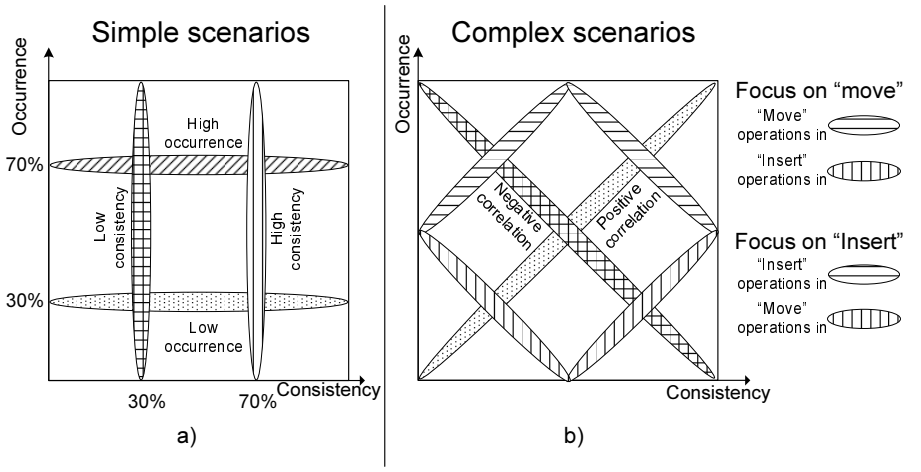


Figure 7.12: Space coverage using different scenarios

4. **High Consistency:** Consistency of activities is kept at 70% while their occurrence varies from 0 to 80%.
   In order to better cover value space, we have designed four additional scenarios (cf. Fig. 7.12b.)
5. **Positive Correlation:** Consistency of a particular activity is positively correlated with its occurrence; i.e., if activity a_j has high occurrence it also has high consistency. This is applied to moved as well as inserted activities.
6. **Negative correlation:** Consistency of an activity is negatively correlated with its occurrence. If a_j has high occurrence value it has low consistency. This is applied to moved as well as inserted activities.
7. **Focus on "move":** High consistency is assigned to moved activities while low consistency is assumed for the inserted activities.
8. **Focus on "insert":** High consistency is assigned to the inserted activities while low consistency is assumed for the moved activities.

We do not only apply these scenarios to better cover value space, but also to analyze whether insert and move operations are considered being equally important by our algorithm. Since insert and move operations are used with same weight when generating our dataset, our heuristic mining algorithm should not show significant differences in respect to these two change patterns when discovering the reference model. Table 7.3 summarizes the parameter settings for different parameters. The technical details for implementing the different scenarios are out of the scope of this thesis (see a technical report for details [102]). In general, we can generate each dataset group by adjusting the values of the different configuration parameters. Since we have 3 options for Parameter 1, 3 options for Parameter 2, and 8 scenarios to cover value space of occurrence and consistency, we generate in total $3 \times 3 \times 8 = 72$ dataset groups.

Parameter 1 (Size of Process Models)	
1. Small-sized	Reference model contains 10 activities
2. Medium-sized	Reference model contains 20 activities
3. Large-sized	Reference model contains 50 activities
Parameter 2 (Similarity of Process Variants)	
1. Small-change	10% activities are changed
2. Medium-change	20% activities are changed
3. Large-change	30% activities are changed
Parameter 3 (Activity Occurrence) and **Parameter 4** (Activity Consistency)	
1. Low Occurrence	Occurrence of activities at 30%; their consistency varies from 0 to 80%
2. High Occurrence	Occurrence of activities at 70%; their consistency varies from 0 to 80%
3. Low Consistency	Consistency of activities at 30%; their occurrence varies from 0 to 80%
4. High Consistency	Consistency of activities at 70%; their occurrence varies from 0 to 80%
5. Positive Correlation	Consistency of an activity is positively correlated with its occurrence
6. Negative correlation	Consistency of an activity is negatively correlated with its occurrence
7. Focus on "move"	High consistency for moved activities; low consistency for inserted ones
8. Focus on "insert"	High consistency for inserted activities; low consistency for moved ones

Table 7.3: Summary of parameter settings

7.5.4 Simulation Setup

For each one of the 72 dataset groups, we apply our heuristic algorithm in order to discover a new reference model by mining the collection of variants. We do not set any constraints on search steps, i.e., the algorithm only terminates if no better model can be discovered. We use a Dell Latitude Laptop (2.4 GHZ CPU and 3.5 GB RAM) to run our simulation under Windows. The following information is documented for each group:

1. **Original reference model**, i.e., the model based on which we perform the changes (cf. Section 7.5.1).
2. **Variant Models**. Based on a given reference model, we generate each variant by configuring the reference model according to the different scenarios described in Section 7.5.3. For each group, we generate 100 variant models. Note that although the 100 variants are generated based on the same scenario, derived models are *NOT* the same. The scenario only describes the probablistic features of the process variants, but not a particular variant.
3. **Search results**. We document both intermediate process models and the final result obtained from our heuristic search. Corresponding change operations are documented as well. As example consider Fig. 7.9. It shows the heuristic search result we obtain when mining the variants from Fig. 7.2.
4. **Fitness and average weighted distance**. Similar to the evaluation results presented in Table 7.2, we compute the values of fitness and average weighted distance for each intermediate process model obtained during the iterations of our heuristic mining algorithm. We additionally document delta-fitness and delta-distance in order to examine the influence of different change operations.
5. **Execution time** of our heuristic search algorithm.

7.6 Simulation Results

While Section 7.5 has described the setup of our simulation and related research questions, this section addresses simulation results in respects to the four research questions in detail.

7.6.1 Basic Performance Analysis

7.6.1.1 Improvement on Average Weighted Distances

For 60 of the 72 groups, we are able to discover a reference model that is different from the original one. Average weighted distance of the newly discovered reference model is *0.765* less than for the original reference model. When compared to average weighted distance of the original reference process model, we obtain a reduction of *17.92%*.

7.6.1.2 Number of Change Operations

For 60 of the 72 groups (i.e., the groups for which we are able to discover a different reference model) we perform in total *284* change operations (i.e., on average *4.73* change operations per group). These 284 change operations comprise *132* insert operations and *152* move operations; i.e., there is *no significant difference* between the number of insert and move operations. This indicates that the two operations are treated in a balanced way by our algorithm. Reason is that we perform on average same number of insert and move operations when generating a dataset group (cf. Section 7.5.3) and such trend is also shown during the discovery of the reference model.

7.6.1.3 Execution Time

Obviously, the number of activities contained in the variant models can significantly influence execution time of our algorithm; i.e., search space becomes bigger for large models since the number of candidate activities being changed becomes higher as well as the number of blocks becomes higher. We therefore analyze execution time of our algorithm taking into consideration the size of process models. Average execution time is shown in Table 7.4.

	small-sized	medium-sized	large-sized
Model sizes (i.e., # activities)	$10 \sim 15$	$20 \sim 30$	$50 \sim 75$
Average search time (s)	0.184	4.568	805.539
Average # of applied changes	1.83	3.52	8.43
Average search time per change	0.111	1.338	98.992
S.Dev of search time per change	0.015	0.152	15.033

Table 7.4: Average execution time for process models having different size

While it takes only little time to discover the result for a collection of small- and medium-sized models, it takes considerable longer time (on average 805.539 seconds) to find the result for a collection of large-sized models. One reason for this is that the considered variant model become larger; another one is that search steps take longer as well. We need to perform on average 8.43 change operations on the original reference model to discover the end result. Reason is that the variants in these groups are more different from each other than those in small- and medium-sized models.[4] As a consequence, the discovered model can be more different from the original one. We additionally compute the mean and standard deviation of search time per search step. Independent from process models' size, standard deviation of search time (per search step) is around $13.5\% \pm 2\%$ of the average search time. This indicates that performance of our algorithm is quite stable. Though it takes significantly longer when dealing with larger process models, we believe execution time is acceptable considering the complexity of the problem (also when comparing it with other data mining problems [181]).

7.6.2 Correlation of Delta-fitness and Delta-distance

As discussed, one important issue we want to analyze is how the fitness value is correlated with average weighted distance. As discussed in Section 7.5, a fitness value is only a "quick guess" of how close a candidate model S_c is to the collection of variants, i.e., it is not as precise as average weighted distance itself, and can also not be perfectly correlated with the latter measure since its computation is at $\mathcal{NP}$-*hard* level.

Our heuristic search algorithm is a best-first approach, i.e., we search whether we can find a process model with higher fitness value. Therefore, it is more useful to measure how much delta-fitness (i.e., difference between the fitness values before and after change) is correlated with delta-distance (i.e., difference between average weighted distances before and after change). Note that it is the improvement of the fitness value (delta-fitness) that guides search steps (cf. Section 7.4). Another reason why we do not directly analyze correlation between fitness and distance is their difference in value ranges. While fitness of a model has value range $[0, 1]$, average weighted distance has value range $[0, +\infty]$. On the contrary, delta-fitness and delta-distance both have value range $[-1, 1]$. Therefore, considering correlation between delta-fitness and delta-distance is more meaningful in the given context. Similar techniques for evaluating fitness functions are widely used when evaluating other heuristic or genetic algorithms [81].

Since every change operation leads to a particular modification of the process model and consequently creates values for delta-fitness and delta-distance, we obtain 284 data samples. Fig. 7.13 plots these data samples as (delta-fitness, delta-distance)-pairs. For example, consider the search result from Table 7.2. We can obtain four data samples for delta-fitness and delta-distance from it: (0.143,0.9), (0.118,0.7), (0.039,0.6) and (0.009,0.25).

[4]The number of change operations is determined by the model size, i.e., we change 10%,

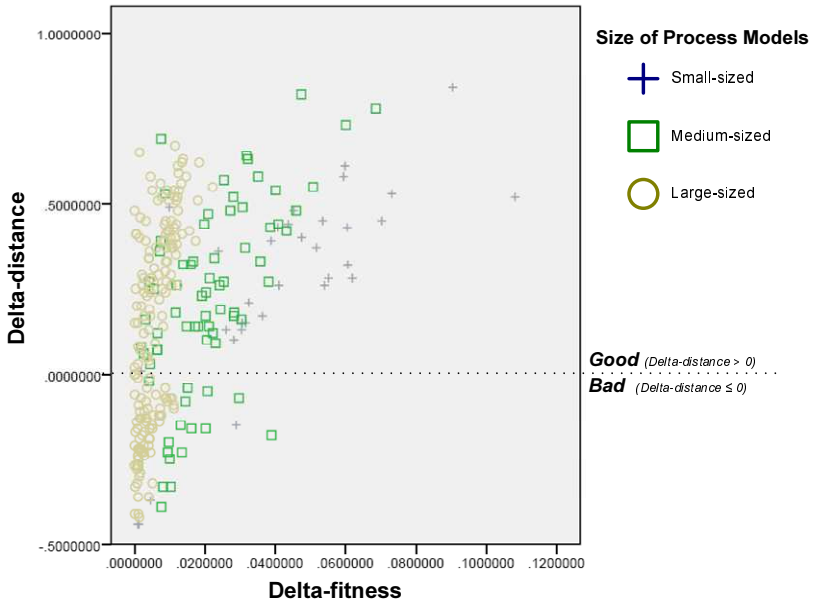


Figure 7.13: Delta-fitness and delta-distance pairs

In Fig. 7.13, the X-axis represents the delta-fitness value and the Y-axis the one for delta-distance. All values for delta-fitness are larger than 0. Note that our algorithm applies a change if and only if it can find a model with higher fitness value. Opposed to this, delta-distance is not always greater than 0. This indicates that sometimes the application of a change operation can make results worse. This is not surprising since fitness value is only a "quick" guess of average weighted distance. We additionally plot a line with delta-distance being 0 to separate "good" samples (positive delta-distance) from "bad" ones (negative delta-distance).

Fig. 7.13 also marks data samples from groups of different model size separately. Obviously, these three groups form three different clusters, i.e., they are not overlapping too much with each other and the larger model size is, the more corresponding data samples position towards the Y-axis. This indicates that process model size can influence delta-fitness value. Reason is that the fitness value of a process model is measured considering all its activities (cf. Formula 7.4). Since a particular operation can only change one activity at a time, its influence on fitness value (delta-fitness) is dependent on the number of activities in the process model. Obviously, the less activities are contained in a process model, the greater the influence of one change operation becomes. Therefore, it is more reasonable to analyze correlation for groups of models with same size.

We use Pearson correlation to measure correlation between delta-fitness and

20% and 30% of activities to configure a variant (cf. Section 7.5.3)

delta-distance [174]. Let X be delta-fitness and Y be delta-distance. We obtain n data samples (x_i, y_i), $i = 1, \ldots, n$. Let $\bar{x}$ and $\bar{y}$ be the means of X and Y respectively, and let s_x and s_y be the standard deviations of X and Y. Then, Pearson correlation can be computed based on Formula 7.6.

$$r_{xy} = \frac{\sum x_i y_i - n\bar{x}\bar{y}}{(n-1)s_x s_y} \tag{7.6}$$

We further test whether correlations are significant, i.e., whether they significantly differ from 0 [174]. Results are summarized in Table 7.5.

	Number of data	Correlation	Probability of $r_{xy} = 0$	Significantly Correlated?
Small-sized	33	0.762	<1.0E-8	Yes
Medium-sized	74	0.589	<1.0E-8	Yes
Large-sized	177	0.623	<1.0E-8	Yes

Table 7.5: Delta-fitness and delta-distance correlations of dataset groups having different model sizes

Obviously, correlations obtained from all three groups are *significant and high*. The high positive correlation between delta-fitness and delta-distance indicates that if we can find a model with higher fitness value, we have high chances to also reduce average weighted distance (i.e., to change the old model to the newly discovered one). A correlation is normally considered being high if it is larger than 0.5 [174]. In our case, all three groups show high correlations, especially when being compared with most other heuristic or genetic algorithms. Note that in most common cases correlations between fitness value and local optimum are low or even negative [81].

7.6.3 Correlation Comparison

In Section 7.6.2 we have discussed correlation between delta-fitness and delta-distance. Since it is important to evaluate performance of our algorithm, we also want to know whether correlation changes when dealing with process models of different size. This is important to know since it directly reflects on the scalability of our algorithm. If correlations is independent from whether we deal with small process models or large ones, scalability of our algorithm will be good (i.e., its performance is stable when dealing with process models of different size.)

When only considering correlations from Table 7.5, it is difficult to derive any trend. Here, lowest correlation value is obtained from the medium-sized models. More important, since we have data samples from three groups, which have different size, the corresponding correlation should have different "credibility". For example, the correlation derived from 177 data samples should be more reliable than the one we can obtain based on 33 samples. To compare correlation val-

ues taking the size of the data sample into account, again we need to employ a statistical approach.

Since the sampling distribution of Pearson correlation analysis is not normally distributed, we first need to perform a *Fisher's Z transformation* to covert Pearson correlation to a normally distributed variable [174]. Let r be a correlation. Then we can perform Fisher's Z transformation as follows:

$$Z(r) = 0.5 \times (\ln(1+r) - \ln(1-r)) \tag{7.7}$$

Distribution of $Z(r)$ has two important properties: First, it is normally distributed. Second, it has a known standard error of $\frac{1}{\sqrt{n-3}}$ where n equals the number of data samples for computing Pearson correlation r. We can compare difference between two correlations r_1 (obtained from n_1 data samples) and r_2 (obtained from n_2 data samples) as follows:

$$\rho(r_1, r_2) = \frac{Z(r_1) - Z(r_2)}{\sqrt{\frac{1}{n_1-3} + \frac{1}{n_2-3}}} \tag{7.8}$$

Approximately, $\rho(r_1, r_2)$ follows standard normal distribution [174]. Table 7.6 shows results that pairwise compare the three groups having different size. In all three comparisons, correlations in Table 7.6 do *NOT* significantly differ from each other; i.e., they are statistically the same. This indicates that performance of our heuristic algorithm does *NOT depend on the size of the models*; i.e., the algorithm can scale up in order to deal with large-sized process models, but without sacrificing its performance.

	ρ value	Probability of being same	Significant?
Small-sized VS. Medium-sized	1.51	0.130	Yes
Medium-sized VS. Large-sized	-0.4	0.689	Yes
Small-sized VS. Large-sized	1.37	0.170	Yes

Table 7.6: Paired correlation comparison

7.6.4 Monotonicity Test

We now test whether our heuristic search algorithm applies the more important change operations (i.e., changes having a higher delta-distance value) at the beginning. For this purpose, we perform two tests. The first one shows whether and to what degree we are able to reduce average weighted distance by only performing a limited number of change operations. In a second test, we want to analyze whether delta-distance is monotonically decreasing.

7.6.4.1 Impact of the Top n% Change Operations

In our simulation we do not control search depth; i.e., our algorithm continues cunning as long as better models can be discovered. As aforementioned, one

important feature of our algorithm is its ability to control how many change operations shall be maximally performed during the discovery procedure. This necessitates that most relevant changes should be performed first. Therefore, we are interested in computing to what degree the first $n\%$ changes contribute to reduce average weighted distance.

As example consider the search results from Table 7.2 (cf. Section 7.4.3). We have performed in total 4 change operations to discover the best model. In total, average weighted distance is reduced from 4.85 (using the original reference model S) to 2.4 (using the discovered model R_4) (cf. Table 7.2). We can analyze how important the changes applied at the beginning are by computing to what degree they have reduced average weighted distance. For example, the first change operation reduces average weighted distance by 0.9. Comparing this with overall distance reduction of 2.45, we can already accomplish $0.9/2.45 = 36.73\%$ distance reduction by performing only the top change operation. Since in the given example there are only 4 change operations, we can claim that by performing the top 25% of the change operations, we can accomplish 36.73% distance reduction. This indicates how important changes applied at beginning are. Clearly, the higher distance reduction is, the more important the respective change is. Similarly, by analyzing delta-fitness, we can compute to what degree fitness improvement can be accomplished by performing only the top changes. Results are summarized in Table 7.7.

	top 33.33% changes	top 50% changes
Fitness gain	57.35%	74.60%
Distance gain	63.80%	78.93%

Table 7.7: Fitness and distance gains when only applying the top changes

In 55 (out of 72) dataset groups, more than 2 change operations are performed during the search. Therefore, we are able to measure monotonicity. Table 7.7 summarizes average distance and fitness gains when applying the top 33.33% and the top 50% change operations. Obviously, changes applied at beginning are *a lot more important* than the ones performed at the end. For example, the top 33.33% change operations contribute 63.80% distance reduction; while the remaining 66.67% change operations only contribute to the remaining 36.20% distance reduction. If we only performed the first half of the change operations, we would already obtain around 80% distance reduction. This simple analysis indicates that changes performed at the beginning are a lot more important than the ones performed at the end.

7.6.4.2 Monotonically Decreasing Score

The approach described in Section 7.6.4.1 is an abstract one, i.e., we analyze the importance of a collection of change operations applied at the beginning. We now analyze whether or not a change operation performed first is really "better" than

the one applied after it (i.e., whether delta-distance is monotonically decreasing when including additional change operations).

Most of the monotonicity tests in data mining or artificial intelligence provide binary answers, i.e., the data sample is either monotonic or non-monotonic [110, 181]. These monotonicity tests are too restrictive in our context since one "problematic" data sample can counteract whole monotonicity test, especially when considering the fact that a heuristic algorithm only enables reasonable "guessing". Restrictive statistical monotonicity test [20] can also not be applied, since on average there are only 4.73 change operations applied per dataset group (cf. Section 7.6.1). This number is too low to conduct any creditable statistical analysis. Therefore, we apply following method for testing monotonicity.

For one dataset group, let S be the original reference model and S' be the discovered one. We obtain $S[\sigma > S'$ with $\sigma = \langle \Delta_1, \Delta_2, \dots \Delta_n \rangle \in \mathcal{C}^*$ being a sequence of changes performed on S during discovery of S'. Let $x_i, x_j (i, j = 1 \dots n)$ be delta-distances of change operations Δ_i and Δ_j respectively. The monotonically decreasing score μ can be computed as follows:

$$\mu = \frac{\left|\left\{(x_i, x_j) | (i < j) \wedge (x_i \geq x_j)\right\}\right|}{n \times (n-1)/2} \tag{7.9}$$

For a given sequence of change operations $\sigma = (\Delta_1, \Delta_2, \dots \Delta_n)$, μ measures the monotonically decreasing score by comparing the delta-distances of every pair of change operations Δ_i, Δ_j $(i, j = 1 \dots n)$. If for two change operations Δ_i, Δ_j with $i < j$ (i.e., change operation Δ_i is performed before change operation Δ_j), delta-distance x_i is larger than or equal to x_j, we consider change operations Δ_i and Δ_j as monotonically decreasing. The monotonically decreasing score μ is then measured by counting how many pairs of change operations are monotonically decreasing. As example consider the results in Table 7.2. For every pair of change operations Δ_i and Δ_j, $i < j$, corresponding delta-distance x_i is larger than x_j. For this case, we obtain $\mu = 1$, i.e., delta-distance of the change operations is monotonically decreasing.

Following this design, every change operation is compared with the ones listed after it in the sequence of applied changes. This way, the change operations performed at the beginning have more influence on μ than the ones performed at the end.[5] Obviously we obtain $\mu \in [0, 1]$: If $\mu = 1$ holds, delta-distance will be monotonically decreasing; if $\mu = 0$ holds, delta-distance will be monotonically increasing. The higher μ is the more delta-distances will be monotonically decreasing. Similarly, we can test monotonicity of delta-fitness. Results are summarized in Table 7.8.

Besides average monotonic decreasing score μ, Table 7.8 shows average monotonic decreasing score μ tolerating an error rate of 5%, i.e., if $i < j$ and $x_i \geq x_j \times (1 - 0.05)$, we still consider x_i and x_j as monotonically decreasing just to avoid

[5] Such feature is also employed in other data mining evaluation techniques (e.g., precision, recall [181]).

	average μ	average μ with 5% error rate
Fitness	0.9942	0.9987
Average weighted distance	0.6682	0.6858

Table 7.8: Average monotonic decreasing score μ as obtained from the simulation

rounding errors. It becomes clear that delta-fitness is almost perfectly monotonically decreasing while such trend in respect to delta-distance is not very strong. This difference is due to the fact that correlation between these two values is not perfect (i.e., it equals 1). Therefore, we cannot claim that delta-distance keeps decreasing as the search continues.

In summary, monotonically decreasing of delta-distance is only *strong at high abstraction level*, e.g., the top 1/3 change operations might accomplish around 2/3 of distance reduction (cf. Table 7.7). However, it is *not very strong*, when comparing individual change operation with each other i.e., we cannot claim that for all cases a change operation performed first is always better than the one (directly) following it (cf. Table 7.8).

7.6.5 Influence of the Different Parameters on our Algorithm

When configuring our datasets we consider the aforementioned parameters: size of process models, similarity between process models, activity consistency and activity occurrence (cf. Section 7.5). In this section, we analyze the influence of these parameters on the overall performance of our algorithm.

For a particular dataset group, we discover its new reference model by stepwise performing a number of change operations on the original one. Consequently, we obtain a collection of data samples in the form (delta-fitness, delta-distance) (cf. Fig. 7.13), of which some might be "bad" ones having negative delta-distance value (cf. Fig. 7.13). Clearly, the less "bad" samples are, the better our algorithm performs. To quantitatively evaluate this performance, we use π as ratio of "good" samples to indicate its performance. When considering the results from the 72 dataset groups, we obtain on average $\pi = 65.14\%$.

We therefore divide the 72 dataset groups based on the (finite) value ranges of the four parameters. Since Parameter 1 (size of process model) has three possible values (i.e., small-size, medium-size and large-size), for example we can divide the dataset groups into three clusters. Each cluster then contains 24 dataset groups having same process models size. We compute mean and standard deviation of π in each of the three clusters. Similarly, we can divide the dataset groups based on the value ranges of the other parameters. Results are summarized in Fig. 7.14.

We first analyze the influence of the size of process models on our algorithm. When measured it using average of π, we can obtain that performance decreases as the size of process model increases. We discuss the reasons for this and also how we can improve performance of our algorithm in Section 7.6.6. In general, delta-

Divided by Size				Divided by Similarity			
	Small	Medium	Large		Small	Medium	Large
Mean	0.843	0.804	0.600	Mean	0.740	0.773	0.719
S. Dev	0.336	0.248	0.314	S. Dev	0.371	0.272	0.312
# of Data	18	21	21	# of Data	18	21	21

Divided by Occurrence and Consistency Scenarios								
	Positive correlated	Focus on "move"	Focus on "insert"	Negative correlated	Low occurrence	High occurrence	Low consistency	High consistency
Mean	0.923	0.766	0.743	0.581	N/A	0.706	0.513	0.945
S. Dev	0.117	0.346	0.331	0.426	N/A	0.324	0.237	0.090
# of Data	9	9	9	8	0	9	8	8

Figure 7.14: Influence each parameter has on the performance of our algorithm

fitness value can be influenced by the size of process models, and consequently the groups with the size of process model being "large" might contain more data samples with low delta-fitness value. This becomes more clear in Section 7.6.6. In particular, we show that low delta-fitness increases the chance of having a bad data sample.

On the contrary, similarity between process models does not influence performance of our algorithm. In all three groups the average of π is around 0.75. In our simulation, we do not separate parameters *activity occurrence* and *activity consistency* when generating the dataset groups. Instead, we use 8 different scenarios to jointly analyze the influence of these two parameters. Fig. 7.14 presents the mean and standard deviation of π in respect to the 8 scenarios.

Results correspond to our analysis depicted in Fig. 7.11. Our algorithm performs better than average (i.e., we obtain high average π value) when dealing with situations in which activities either have high occurrence as well as high consistency, or low occurrence as well as low consistency. For example, our algorithm performs better than average in scenario "positive correlated", where occurrence is positively correlated with consistency. However, when dealing with the scenarios where activity consistency and occurrence are both not high, performance of our algorithm is less optimal. For example, in scenario "negatively correlated" where occurrence is negatively correlated with consistency, we obtain one of the lowest results.

Regarding a collection of process variants, except Parameter 1 (i.e., the size of process models), which can be easily determined, the other three parameters are costly to compute. In particular, we need to compute distance and bias between the original reference model and every process variant in order to determine the other three parameters. Note that computing these biases and distances has $\mathcal{NP}$-*hard* complexity (cf. Def. 9). In our simulation, we are able to obtain such information only because we have generated the datasets ourselves. In particular, this indicates that for a given collection of process variants, but non-availability of other information, it is difficult to predict how our algorithm can perform.

However, we can roughly estimate the parameters based on the performance (measured by π) of our algorithm. For example, if we can identify a lot of "good" points while only few "bad" points exist during the search, we could expect both activity occurrence and activity consistency being potentially high in the given variant collection. Admittedly, such estimation is not very precise.

7.6.6 Pruning Threshold Training

If we revisit our delta-fitness and delta-distance graph as plotted in Fig. 7.13, it becomes clear that there are still some "bad" data points. Those are the points with positive delta-fitness, but negative delta-distance. Though these bad data points can never be prevented due to the nature of heuristic algorithms, we can at least reduce them, i.e., reduce the chance that they appear.

When analyzing these "bad" points, we can find them most time at the down-left corner of the chart. This indicates that if delta-fitness is low, the chance of getting a negative delta-distance will get bigger. In order to quantitatively evaluate this, we introduce *precision* here. Let X be delta-fitness and Y be delta-distance. We obtain n data samples (x_i, y_i), $i = 1, \ldots, n$. Given a delta-fitness value x, we can compute *precision* $p(x)$ as follows:

$$p(x) = \frac{\left|\left\{(x_i, y_i) | (x_i \geq x) \wedge (y_i > 0)\right\}\right|}{\left|\left\{(x_i, y_i) | x_i \geq x\right\}\right|} \tag{7.10}$$

Given delta-fitness value x, precision $p(x)$ measures the ratio of "good" data samples (with delta-distance being larger than 0) among data samples with delta-fitness value larger or equal to x. The higher $p(x)$ is, the more "good" sample we have. Note that precision is widely used in fields like information retrieval [14] and data mining [181]. Fig. 7.15 depicts precision value $p(x)$ for different values x of delta-fitness.

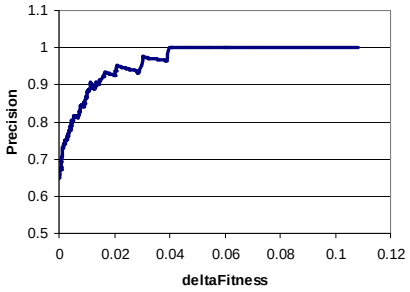


Figure 7.15: Delta-fitness and precision chart

From Fig. 7.15, it becomes clear that, the lower delta-fitness is, the lower precision will be. When considering only data samples with delta-fitness being larger than 0.0401, all corresponding delta-distance values are positive, i.e., all

them are all "good" samples. Precision keeps reducing until 65.14% when considering all data points (i.e., $x = 0$). This indicates that many "bad" data samples occur with low delta-fitness values.

Precision analysis indicates that we can probably improve our heuristic mining algorithm by determining a threshold value for delta-fitness. Since most of the "bad" data samples are obtained with low delta-fitness values, we only allow performing a certain change in case delta-distance value is larger than this threshold value. In the following we introduce two approaches for discovering such threshold based on our simulation data.

7.6.6.1 Classification Tree

We first introduce an approach using classification trees [138]. By learning from the "good" and "bad" data samples, we should be able to classify them by a threshold delta-fitness value. Let X be the delta-fitness and Y be the delta-distance. We obtain n data samples (x_i, y_i), $i = 1, \ldots, n$. To each data sample we can assign a binary value z_i being "$TRUE$" or "$FALSE$" depending on the value of y_i. If $y_i > 0$ holds, z_i is set to $TRUE$, otherwise, z_i is set to $FALSE$. We therefore can build a classification tree using delta-fitness x_i and binary variable z_i. We choose algorithm *C4.5*, which is one of the most popular classification algorithms to build the classification tree [138], and use *Weka*, which is one of the most popular open-source data mining tools, to compute the result [228]. Details about this classification algorithm and the data mining tool are out of the scope of this thesis. However, note that these are standard methods to perform such a classification. The resulting classification tree is shown in Fig. 7.16.

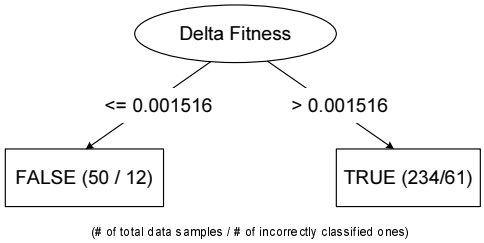


Figure 7.16: The classification tree build based on delta-fitness values

The simple classification tree has divided the data samples into two groups based on the values of delta-fitness. When delta-fitness x_i of a certain data sample (x_i, y_i) is less or equal 0.001516, the classification tree will classify it as $FALSE$, i.e., it is more likely to be a "bad" data sample with delta-distance lower than 0. On the contrary, if x_i is larger than 0.001516, it is more likely to be a "good" data sample. The classification tree is also not 100% precise: 12 out of 50 data samples with delta-fitness lower than 0.001516 actually provide positive delta-distance while 61 out of 234 data samples with delta-fitness larger than 0.001516 provide negative delta-distance values. Though the classification

tree is not perfectly precise, it is one of the best ones we can build purely based on the delta-fitness value.

The discovered threshold 0.001516 therefore can help us to improve performance of our heuristic algorithm. More precisely, we should only apply changes for which improvement of the fitness value is larger than the given threshold. For this case, we expect reducing the chance of performing a wrong change (a change which even enlarges average weighted distance between discovered model and variants (negative delta-distance)). If we filter the data sample using this threshold value for delta-fitness, we can increase precision of the whole data sample from 65.14% to *73.93%* and further reduce average weighted distance. More precisely, if we do not set any threshold average weighted distance per group is reduced by 0,765 (17.92% reduction compared to the original reference model). By contrast, if we set the above mentioned threshold, we can reduce average weighted distance by *0.879* (20.84% distance reduction compared to the original reference model); i.e., using such threshold can lead to better search results (i.e., higher distance reduction). In summary, we can discover better models by setting a threshold value to guide our heuristic search.

7.6.6.2 Determining Threshold Based on Overall Distance Gain

In Section 7.6.6.1, we have discussed a standard data mining approach to improve our algorithm, while in this subsection we introduce a more intuitive and straightforward approach to discover a threshold value. One disadvantage of the above mentioned classification tree is that it cannot consider importance of a "good" or a "bad" data sample. The threshold is trained by a binary decision variable z_i which is either $TRUE$ or $FALSE$. It does not consider how much $TRUE$ or how much $FALSE$ by directly measuring average weighted distance value. Here, we introduce a method to discover a threshold by considering delta-distance value. Let X be delta-fitness and Y be delta-distance. We obtain n data samples (x_i, y_i), $i = 1, \ldots, n$. Given a delta-fitness value x, we can compute $\eta(x)$ as follows:

$$\eta(x) = \sum_{x_i \geq x, i=1}^{n} y_i \tag{7.11}$$

$\eta(x)$ measures the sum of the delta-distance values after filtering out those data samples with delta-fitness being lower than a given threshold x. Regarding our dataset, Fig. 7.17 depicts the curve of $\eta(x)$ according to the value of x. We specially zoom the part with x being contained in interval [0, 0.006].

$\eta(x)$ keeps increasing as x decreases. This is easy to understand since the lower x is, the less data samples will be filtered out. However, $\eta(x)$ reaches its maximum of 51.72 when x equals 0.0014, and starts to decrease as x decreases. This indicates that in interval [0, 0.0014) of delta-fitness, there are more data samples with negative delta-distance. If 0.0014 is used as threshold value to guide our heuristic search (i.e., we only perform a change if the improvement

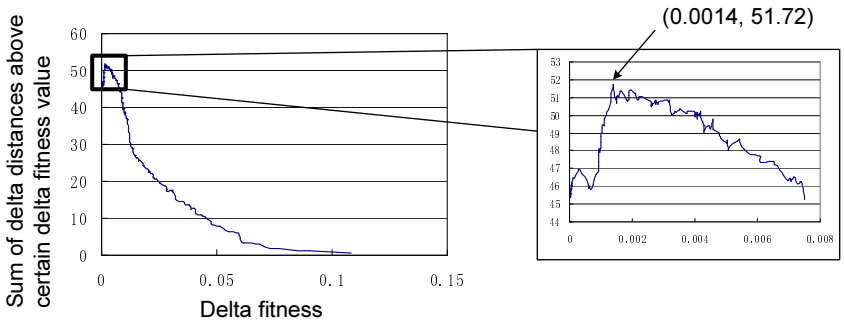


Figure 7.17: Delta-fitness x and its corresponding $\eta(x)$

of fitness value is larger than 0.0014), we can further reduce average weighted distance from 0,765 (17.92% reduction compared to the original reference model) per group to *0.892* (20.85% distance reduction compared to the original reference model) per group. This indicates that using such threshold can lead to better search results (i.e., higher distance reduction). The corresponding precision will increase from 65.14% to 73.22%.

Note that the threshold value obtained by our simulation data is a case-specific one. It cannot be generalized that using these threshold values always improves performance of our algorithm. However, it is useful to perform such analysis in the context of this thesis in two respects. First, we indicate that for lower delta-fitness value the algorithm bears higher chance of performing a wrong change. Consequently, the user should be careful when discovering a model with only slightly better fitness value. Second, using the suggested approaches, a user can obtain their own threshold value based on their own dataset. We present the corresponding training method in this thesis just to provide a guideline on how to apply and/or improve our heuristic mining algorithm to a domain-specific field, i.e., user can obtain a threshold value to make our heuristic mining algorithm work better in context of their specific problem.

7.7 Summary

This chapter provided a heuristic search algorithm that supports the discovery of a block-structured reference process model by learning from a collection of block-structured process variants. Adopting the discovered model as new reference process model will make process configuration easier; i.e., it will require less efforts to configure these variants (measured in terms of the number of required change operations). In particular, our heuristic algorithm can also take the original reference model into account such that the user can control to what degree the discovered model is different from the original one. This way, we cannot only *avoid spaghetti-like process models* but also control how many changes we want

to perform. Our algorithm can automatically determine which activities shall be considered in the reference model. Filtering or pre-analysis of the activity sets are not needed in this context.

We evaluated our algorithm by performing a comprehensive simulation. Based on our simulation results we can draw the following conclusions:

1. The fitness function of our heuristic search algorithm is correlated with the average weighted distance with a high correlation value. This indicates good performance of our algorithm since the approximation value we use to guide our algorithm is nicely correlated to the real one.
2. Our algorithm can scale up. Its performance, which is measured based on the correlation between fitness and distance, is independent from the size of the process models.
3. When discovering the new reference model by changing the original one, the more important changes, which largely reduce average weighted distance to the variants, are performed at the beginning. Our simulation results indicate that the first 1/3 of the applied changes result in about 2/3 of overall distance reduction.

We also discussed two approaches for further improving the performance of our heuristic search algorithm by learning an appropriate threshold value. Though the results must not be generalized to all cases, the suggested approach can also support users to adapt our algorithm to a domain-specific context. In Chapter 9, we will discuss how we successfully applied our heuristic algorithm to a real-world scenario.

Part III

Validation & Discussion

8 Algorithm Comparison

8.1 Introduction

In Chapters 6 and 7 we have introduced two algorithms for discovering a reference process model based on a given collection of process variants. Each algorithm has its specific properties and is particularly suitable for a certain scenario. In this chapter we compare the two algorithms qualitatively as well as quantitatively. In particular, we aim at answering our fifth research question (cf. Section 1.2):

What are characteristic properties of the solution approaches we propose for supporting process variant mining? Under which circumstances is one approach better suited than the other?

The remainder of this chapter is organized as follows. Section 8.2 describes the proof-of-concept implementation of our algorithms. Section 8.3 then provides the results of a qualitative as well as quantitative comparison between the developed algorithms. We systematically compare our algorithms with traditional process mining techniques from both a behavioral and structural perspective in Section 8.4. Finally, Section 8.5 concludes with a summary.

8.2 Proof-of-Concept Prototype

Figure 8.1 depicts the architecture of our system for process variant mining. We apply the ADEPT2 Process Template Editor [148, 144, 143] for creating and visualizing process variants. For each process model, the editor can generate an XML (eXtensible Markup Language [234]) representation with all relevant information (like nodes, edges, blocks) being marked up. We store created variants in a variant repository (cf. Fig. 8.1) which can be accessed by our mining algorithms. According to the XML schema of the process editor, we use the `Model Reader` component to transform a process model into its order matrix and we apply the `Model Writer` component to transform an order matrix back into a process model (cf. Chapter 4).

Based on order matrices, we can measure the distance between two process models (cf. Chapter 5). We can further use order matrices when mining process

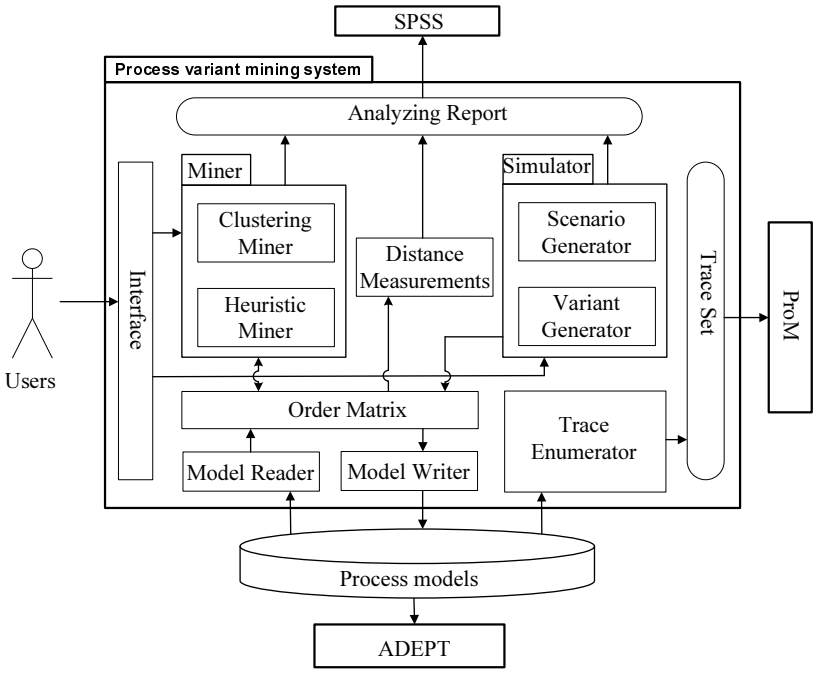


Figure 8.1: Architecture of process variant mining system

variants. In this context, we implemented and tested both the clustering algorithm (cf. Chapter 6) and the heuristic algorithm (cf. Chapter 7) using Java. The models we obtain when applying our algorithms is stored in the variant repository.

Through `Interface` users can set parameters for our mining algorithms (e.g., search limitations for our heuristic algorithm). Further, they can configure the `Simulator` component. The latter has been designed for simulation purposes (cf. Chapter 7) and is used to test the performance of our algorithms. The `Simulator` component consists of the two sub-components `Scenario Generator` and `Variant Generator`. We use the former to configure the parameters of each dataset group (cf. Section 7.5) and apply the latter for configuring the particular variants. The generated process variants are stored in the repository and can be analyzed at any time by applying our mining algorithms.

After running the `Miner`, `Simulator` or `Distance Measurement` components, the obtained results are stored in `Analyzing Report`. Depending on which function we call, this report contains information like average weighted distance of the discovered process model or delta-fitness and delta-distance after the application of each change operation suggested by our heuristic algorithm. The report is organized in a way that it can be directly accessed by statistical analysis software. In the context of this thesis we applied the SPSS [179] package, which is one of the most popular tools for statistical analysis. In this way, we can directly per-

form analysis like the Pearson correlation test (cf. Section 7.6) when evaluating performance or testing the properties of our algorithms.

In addition, we implemented a `Trace Enumerator` which can enumerate all traces producible by a process model (we discuss details in Section 8.4). The enumerated traces are formated according to MXML[1] [64, 201] which can be directly accessed by ProM [201]. ProM is one of the most popular open source tools for process analysis which covers a large spectrum of topics and particularly focuses on process mining [195]. By using ProM we can compare our algorithms with well-known process mining algorithms (cf. Section 8.4) to discover the pros and cons of the different algorithms. The connection to ProM further provides the foundation for integrating our algorithms with process mining algorithms so that we can nicely balance between structure and behavior [61, 62]. Indirectly, we can connect ADEPT, which is one of the most mature process management systems supporting dynamic process changes (cf. Section 2.3), and ProM, which is one of the most popular tools fostering process analysis and process mining.

We believe that the presented process variant mining system provides an important step forward towards full process life cycle support for dynamic processes [213].

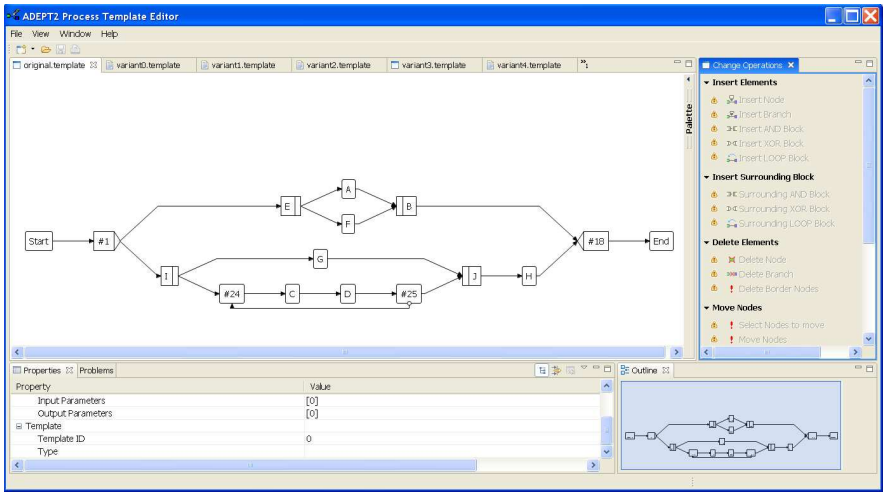


Figure 8.2: Screenshot of the prototype

8.3 Comparing the Algorithms for Process Variant Mining

In the following, we systematically compare the two algorithms we developed.

[1] MXML is the input format of ProM.

8.3.1 Qualitative Comparison

Inputs and Goals. Fig. 8.3 illustrates how our heuristic mining algorithm differs from the clustering one in respect to goals and inputs. Fig. 8.3 represents each process variant S_i as single (white) node in the two dimensional space. Our heuristic algorithm tries to discover a new reference process model by performing a sequence of change operations on the original reference model. In particular, it balances two "forces": one is to bring the new reference model S_c closer to the variants (i.e., to the bull's eye S_{nc} at the right); the other force is to not "move" it too far away from the original reference model S, i.e., S_c should not differ too much from S. Our heuristic algorithm provides such flexibility by allowing process engineers to set a maximum search distance. Our simulations (cf. Section 7.6) showed that the change operations that are applied first to the (original) reference model are more important than the ones positioned at the end; i.e., they reduce the distance between reference model and variants more. Consequently, when setting maximal search distance to filter out less important changes at the end, we do not influence overall distance reduction too much.

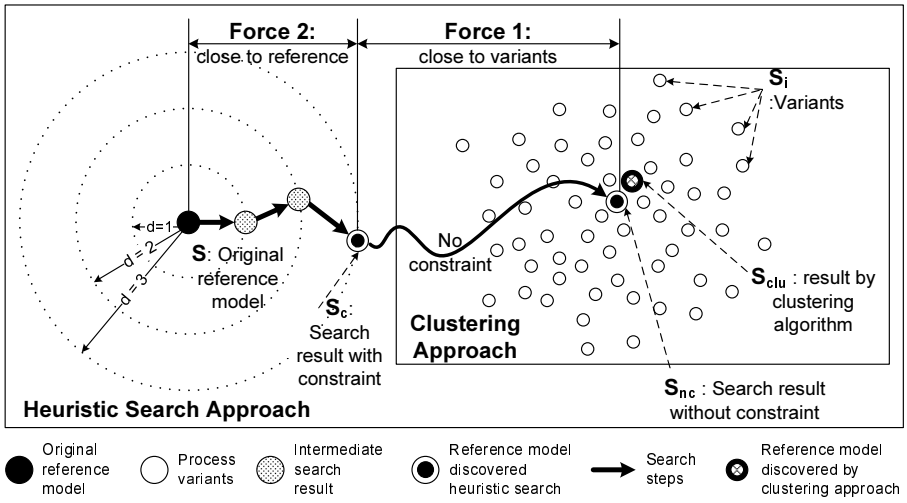


Figure 8.3: High-Level overview of the two algorithms

While the above scenario presumes knowledge of the original reference model (Scenario 2 in Fig. 1.3), we also must cope with cases in which there exists only a collection of process variants, but no original reference process model is known (Scenario 1 in Fig. 1.3). The goal of our clustering approach therefore purely is to discover the "center" of the variants, i.e., a reference process model with shortest average weighted distance to the variants. In particular, no knowledge about the original reference model is required. In principle, it is also possible to apply our heuristic algorithm to this scenario. We just need to start with an "empty" model

S and do not set any search limitation.[2] However, since we do not need to balance the two forces and to perform the important change operations at the beginning of the search, the clustering algorithm is expected to be faster and to provide additional information on the search result. We discuss this quantitatively in Section 8.3.2.

Design Principles. Our heuristic algorithm discovers a better reference model by applying a sequence of change operations to the original one. To enable quick decisions on a large search space, we use a fitness function (cf. Section 7.3) to evaluate how well a candidate model fits to the variants. This fitness function only provides a global evaluation, but does not show how good each part of the candidate model fits to the variants. On the contrary, the clustering algorithm discovers a reference process model by enlarging blocks. By evaluating separation and cohesion, we are able to determine how well each part of the discovered reference model fits to the variants; i.e., due to its different design the clustering algorithm returns more information than the heuristic one.

Complexity of the two algorithms differs as well. Despite polynomial complexity of computing the fitness of a candidate model, worst case, enumerating all blocks in a candidate model has 2^n complexity, where n corresponds to the number of nodes in its process structure tree.[3] On the contrary, our clustering algorithm has polynomial complexity since computing separation and cohesion is both polynomial. To be more precise, if n corresponds to the number of activities and m to the number of variants, complexity of the clustering algorithm is $\mathcal{O}(n^2m + n^3)$. This implies that the clustering algorithm can quickly compute the reference process model of a large collection of process variants, while the heuristic algorithm may take considerable longer. We present a quantitative comparison in Section 8.3.2.

Pros & Cons. The differences between the two algorithms are summarized in Table 8.1. Additional attention should be paid to their pros & cons. Since the clustering algorithm has polynomial complexity, it runs significantly faster than the heuristic algorithm. Using *Separation* and *Cohesion*, we obtain information about how each part of the discovered reference process model fits to the variants. However, our clustering algorithm cannot control the discovery procedure or distinguish important changes from less relevant ones as our heuristic algorithm does. Based on our illustrative example, our clustering algorithm can discover same process model (cf. Fig. 6.8) as our heuristic algorithm (cf. Fig. 7.9) when choosing the threshold value from interval [0.6,0.8). In many other

[2]Here, we have the assumption that the process model, based on which our heuristic algorithm starts, does not influence the search result. We will systematically test this property in our future work.

[3]This worst-case scenario will only occur if any combination of activities may form a block (like a process model for which all activities are ordered in parallel to each other). In this case, the number of blocks containing one activity is C_n^1, the number of blocks containing two activities is C_n^2, and the number of blocks containing $m(m \leq n)$ activities is C_n^m. When summing them together, we obtain in total $\sum_{m=1}^{n} C_n^m = 2^n$ blocks. However, during our simulations, in most cases we were able to enumerate all blocks of a process model within milliseconds. This indicates good performance in practice.

cases the discovered model was less optimal for the clustering algorithm since its search space is considerably smaller.

	Clustering Algorithm	**Heuristic Algorithm**
Input	Collection of process variants.	Collection of process variants + original reference process model
Goal	Discover reference process model with shortest average weighted distance to the variants	Discover better reference process model with maximum distance to the original one
Use cases	Scenario 1 (cf. Section 1.2).	Scenario 2 (cf. Section 1.2).
Design principle	Local view: Discover reference process model by enlarging blocks	Global view: Discover reference process model by searching for better candidate models
Complexity	$\mathcal{O}(m^2n + m^3)$ (m : # activities; n : # variants)	Sub-steps contain $\mathcal{NP}$-*hard* problems
Pros & Cons	1. Runs very fast 2. Provides local view on how each part of the reference process model fits to the variants 3. Activity set can be flexibly chosen by user	1. Automatically selects the activity set 2. Can control the maximum distance between the original reference process model and the discovered one 3. Applies more important changes at the beginning

Table 8.1: Qualitative comparison between clustering and heuristic algorithms

8.3.2 Quantitative Comparison

We now compare our two algorithms quantitatively by analyzing how fast they run and how good the discovered models are. We use the same data for this comparison as for the evaluation of our heuristic algorithm (cf Section 7.5).

We generated 72 groups of datasets representing different scenarios. Each group contains 1 reference process model and 100 process variants (cf Section 7.5). Based on this, with each algorithm we discovered a new reference process model. We documented execution time as well as the average weighted distance between the resulting reference process models and the variants. Results are summarized in Table 8.2, which indicates that the clustering algorithm runs significantly faster than the heuristic one. However, the reference model discovered by the heuristic algorithm has shorter average weighted distance to the variants than the one discovered by the clustering algorithm. On average, the average weighted distance of the model discovered by the heuristic algorithm, is about 81% to 87% of the one discovered by the clustering algorithm.

	Average execution time			Average weighted distance
	# of activity per variant	Clustering Algorithm	Heuristic Algorithm	Heuristic algorithm / Clustering algorithm
Small-sized	10 - 15	0.013	0.184	86.25%
Medium-sized	20 - 30	0.022	4.568	87.11%
Large-sized	50 - 75	0.181	805.539	81.03%

Table 8.2: Performance comparison between clustering and heuristic algorithm

8.4 Comparison with Existing Process Mining Algorithms

Process mining has been extensively studied in literature. As discussed in Section 2.5, its key idea is to discover a process model by analyzing *execution behavior* of process instances as captured in execution logs. The latter typically document the start/end of each activity execution, and therefore reflect behavior of implemented processes. As further discussed in Chapter 3, process mining techniques can be applied in our context as well. Consider our example from Fig. 7.8. For each process variant S_i, we can first obtain its trace set $\mathcal{T}_{S_i}$ by enumerating all traces producible on S_i [230]. If a process model contains loop structures (i.e., it can generate an infinite number of traces) we assume that a loop-block is executed either once or twice. Despite this simplification, the number of traces producible by a process model can still be extremely large; e.g., if a parallel branching compromises five branches, of which each contains five activities, the number of producible traces is $(5 \times 5)!/(5!)^5 = 623360743125120$. This explains why we conduct the comparison only in small scale.[4]

The trace sets generated for the variants are merged into one trace set $\mathcal{T}$ taking the weight of each variant into account. As example consider Fig. 7.2. As S_1 accounts for 25% of the variants, we ensure that each trace producible by S_1 has the same number of instances and the sum of all instances producible by S_1 accounts for 25% of the instances in $\mathcal{T}$ as well. We consider $\mathcal{T}$ as execution log since it fully covers the behavior of the given variant collection.

Since all activities captured in an execution log will be included in most the discovered process model by process mining algorithms (same as our clustering

[4]Note that the main goal of process mining algorithms is to discover a process model based on the traces captured in the execution log (cf. Section 2.5). In most cases, an execution log only captures a small fraction of the traces producible by the underlying process model [195]. This means that process mining techniques do not really require enumerating all traces producible by a process model for further analysis. In the context of our algorithm comparison, we decide to enumerate all traces producible by each variant for the following two reasons:

1. In the context of our research, we do not assume the existence of an execution log. However, most process mining evaluation criteria rely on traces, e.g., *fitness, successful execution*, *proper completion*, and *behavioral appropriateness* (cf. Section 8.4.1). Therefore, we adopted a general approach to first enumerate all traces and then to discover a process model. This way, we were able to compare all related mining algorithms based on the same approach.

2. As an alternative approach, we can randomly enumerate a collection of traces producible by a process model, and consider these random traces as an execution log which partially reflects the behaviors of the process model. One challenge in this context is that we are not aware of any technique which can enumerate a representative set of traces (but not all traces) to express the behavior of a process model. If we purely use a random fraction of traces, we cannot ensure a fair comparison because the results might significantly depend on the randomly selected trace set. Therefore, we decide to enumerate all traces producible by a process model. In this way, we ensure that all possible behavior is considered, and the results are not influenced by randomness.

algorithm), we introduce two additional datasets. In the first one, we filter out all activities a_j whose activity frequency $g(a_j)$ is lower than 0.2 in the variant collection (cf. Def. 12); e.g., in our example activity Z in S_5 as well as silent activity τ (representing the loop in S_4) are ignored (cf. Fig. 7.2). For this extended data set, we can determine trace set $\mathcal{T}_{0.2}$. In the second additional dataset, we filter out activities whose activity frequency is lower than 0.6. Regarding our example, besides Z and τ, we then additionally filter out activity Y in S_2, S_3, S_5, and S_6 (cf. Fig. 7.2). Consequently, we obtain trace set $\mathcal{T}_{0.6}$ which contains all traces producible by the reduced variants. Note that $\mathcal{T}_{0.6}$ has same activity set as the model discovered by our heuristic algorithm (cf. R_4 in Fig. 7.9). The enumerated trace sets $\mathcal{T}$, $\mathcal{T}_{0.2}$ and $\mathcal{T}_{0.6}$ are imported into the ProM framework. In our comparison, we consider alpha algorithm [197], heuristic miner [221], genetic algorithm [39] and multi-phase miner [200]. These are well-known algorithms for discovering process models from execution logs. Fig. 8.4 shows three Petri Net models discovered by different algorithms based on different data sets. The enumerated trace sets $\mathcal{T}$, $\mathcal{T}_{0.2}$ and $\mathcal{T}_{0.6}$, as well as the process models discovered by different algorithms based on them are available at *http://wwwhome.cs.utwente.nl/~lic/Resources.html*.

8.4.1 Evaluation Criteria

Our algorithms focus on the *structural* perspective of process models, i.e., our goal is to configure the variant models out of a reference model with minimal efforts (i.e., with minimal number of high-level changes). On the contrary, traditional process mining focuses on process *behavior*, i.e., the discovered process model should cover the behavior of the variant models (as reflected by their trace sets) [197, 39, 200, 221]. In the following, we compare our algorithms with existing process mining algorithms from both a structural and a behavioral perspective.

Since most existing process mining algorithms are either based on Petri Nets or EPCs, we transform the process models discovered by the different algorithms either into Activity Nets, Petri Nets or EPCs in order to enable a better comparison (see [40, 230] for respective transformation techniques). Such model transformation enables us to apply existing metrics [168, 113] and tools [201] for process model evaluation instead of introducing new ones. We briefly describe the metrics applied in this context and refer [168, 113, 201] for details. In the following we first introduce three parameters to evaluate the structure of process models, namely *average weighted distance*, *structural appropriateness*, and *# splits/joins in EPC*.

1. **Average weighted distance** measures the efforts to configure the process variants out of the discovered reference process model. The lower this parameter is the easier the variants can be configured (cf. Chapter 5).

2. **Structural appropriateness** measures the complexity of a Petri Net by computing the ratio between labeled transitions and nodes (transitions and

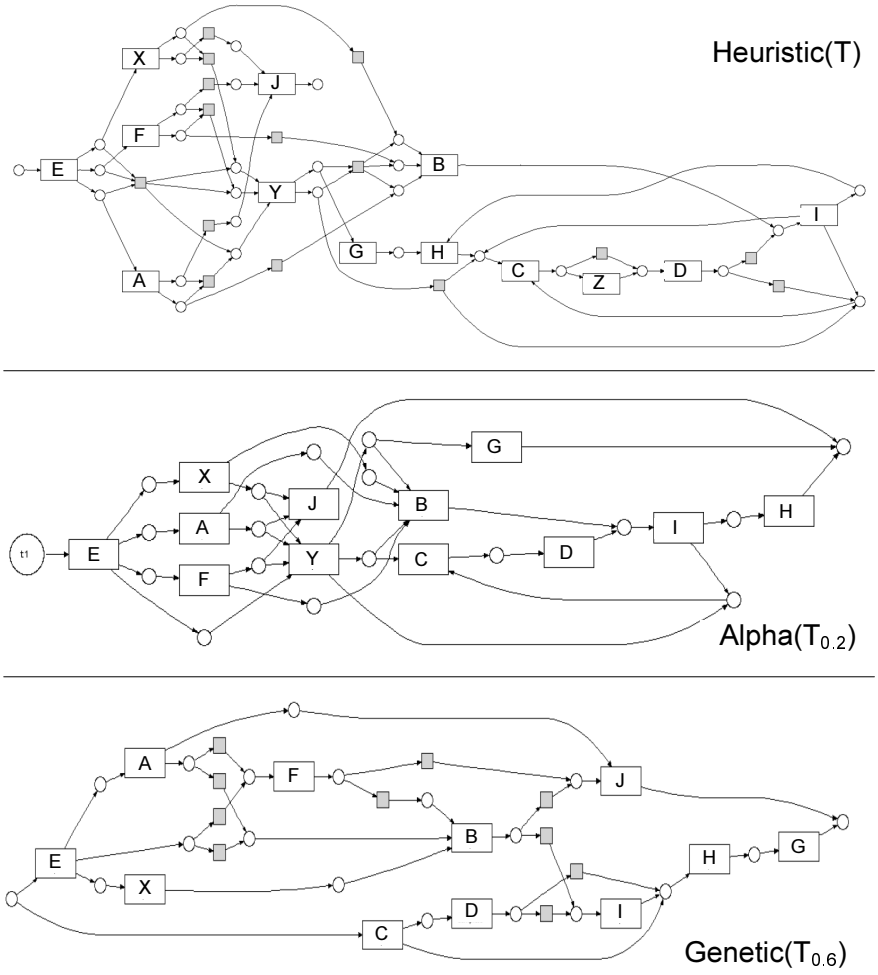


Figure 8.4: Example process models discovered by process mining algorithms

places) [168]. The value range of this parameter is [0,1]; the higher this value is, the simpler the Petri Net will be.

3. **# Splits/Joins in EPC** measures the number of splits and joins as contained in an Event Process Chain (EPC). It can be used to measure the complexity of an EPC [113]. The higher this parameter is, the more choices end users need to make when executing the process model and the more complex the respective EPC will be.

We additionally use four parameters to evaluate the behavior of discovered process models: *behavior fitness, successful execution*, *proper completion*, and *behavioral appropriateness.*

1. **Behavior fitness** evaluates whether the discovered process model, which is represented as Petri Net, complies with the behavior as captured in the execution log [168]. One way to investigate behavior fitness is to replay the log on the Petri net. This is done in a non-blocking way; i.e., if there are tokens missing to fire a transition in the discovered model, they are artificially created and replay proceeds [168]. The value range of this parameter is [0,1]. The higher behavior fitness is, the better the discovered model will cover the given trace set.

2. **Successful execution** measures the percentage of traces in an execution log that can be successfully executed by the discovered process model [168]. The value range of this parameter is [0,1]. The higher it is, the more traces in the execution log can be re-produced based on the discovered model.

3. **Proper completion** measures the percentage of traces in an execution log that lead to proper completion [168]. Compared to "successful execution" this parameter further requires that the analyzed process model reaches an end state after replaying a trace. The value range of this parameter is [0,1]. The higher it is, the more traces from the execution log will result in proper completion.

4. **Behavioral appropriateness** measures to what degree the discovered process model allows "extra" behaviors, i.e., the behaviors which are allowed by the process model but are not observed in the execution log [168]. Similar to behavior fitness, we can evaluate behavioral appropriateness by replaying the log on a Petri Net. The value range of this parameter is [0,1]. The higher it is, the less "extra" behavior is allowed by the discovered process model [168].

8.4.2 Evaluation Results

Our evaluation results are summarized in Table 8.3. To differentiate our heuristic algorithm from the heuristic miner as known from process mining [221], we denote our heuristic algorithm as "Heur. Var." in Table 8.3. Note that *behavior appropriateness* is not included in Table 8.3 because in several cases the conformance checker in ProM cannot measure the behavior appropriateness of the discovered process models[5] in a reasonable time (e.g., within a couple of days).

It is not surprising that both our heuristic and our clustering algorithm are able to discover a process model of simple structure. Independent from which dataset we use, the process models discovered by our two algorithms have better scores for all three parameters relating to the structure of the discovered model;

[5] not only the models resulting by Multi-Phase miner

[6] We manually transformed Petri Net or EPC models into block-structured process models to measure their average weighted distance to the variants. To ensure the resulting models are not overly complex, some trivial behaviors of the Petri Net or EPC models are not considered.

		Structure measurement			Behavior measurement		
Data-set	Algorithms	Average weighted distance[6]	Structural appropri-ateness	# Joins/ splits in EPC	Behavior fitness	Successful execu-tion	Proper com-pletion
$\mathcal{T}$	Heur. Var.	2.4	0.481	6	0.876	0.353	0.353
	Clustering	4.75	0.468	8	0.737	0.120	0.120
	Alpha	8.55	0.441	15	0.646	0	0
	Heuristic	8.85	0.258	31	0.437	0.042	0
	Genetic	6.6	0.341	19	0.811	0.342	0.009
	Multi-phase	2245 arcs, 515 transitions		19	In theory, all equals 1		
$\mathcal{T}_{0.2}$	Heur. Var.	2.4	0.481	6	0.886	0.382	0.382
	Clustering	2.6	0.482	6	0.784	0.133	0.133
	Alpha	6.9	0.466	12	0.706	0	0
	Heuristic	8.2	0.274	12	0.789	0.268	0
	Genetic	5.9	0.424	13	0.846	0.460	0.009
	Multi-phase	1534 arcs, 384 transitions		18	In theory, all equals 1		
$\mathcal{T}_{0.6}$	Heur. Var.	2.4	0.481	6	0.851	0.327	0.337
	Clustering						
	Alpha	6.85	0.5	7	0.814	0.407	0
	Heuristic	7.85	0.462	10	0.736	0.407	0
	Genetic	3.2	0.325	13	0.886	0.394	0.278
	Multi-phase	1266 arcs, 302 transitions		17	In theory, all equals 1		

Table 8.3: Performance comparison with process mining algorithms

i.e., they have lower average weighted distance, higher structural appropriateness, and less number of splits / joins in the corresponding EPC model. Except multi-phase miner [200], none of the algorithms discovered a process model with behavior fitness being 1. Note that multi-phase miner was designed in a way that it always discovers a process model with fitness 1. Despite the fact that the models discovered by multi-phase miner are extremely complex, they also allow for more behavior not covered by the variants [200, 168]; i.e., results are often underfitting.[7] Consequently, we consider the costs of multi-phase miner for reaching behavior fitness 1 as being too high (extremely complex structure and overfitting). When excluding multi-phase miner, evaluation results show that even if we apply traditional process mining algorithms for discovering a process model that covers behavior of the variants best, the resulting model might NOT be able to support all kinds of behavior as captured by the variants. This indicates the necessity for process configuration: i.e., it is not sufficient to maintain only one model which covers all behavior; instead we must enable process configurations at both run-time and build-time to obtain different process variants which support specific behaviors in different scenarios.

For the given dataset, the behavior measurements of the process model we discovered by our heuristic algorithm are also very good. Note that our heuristic algorithm discovers the same model (cf. R_4 in Fig. 7.9) based on either $\mathcal{T}$, or

[7] In principle, it is possible to measure underfitting using parameter like *behavioral appropriateness* [168]. However, due to the complexity of the discovered models, the conformance checker in ProM cannot measure some of discovered models in a reasonable time.

$\mathcal{T}_{0.2}$ or $\mathcal{T}_{0.6}$. This model has highest behavior fitness for trace sets $\mathcal{T}$ and $\mathcal{T}_{0.2}$, and only a few percent less than the genetic algorithm for $\mathcal{T}_{0.6}$. This evaluation results imply that the behavior aspect of the discovered model has not been sacrificed that much. However, this was rather unexpected because our heuristic variant mining algorithm is focusing on structure rather than on behavior. As example consider the examples we presented in Section 3.3, which compare the goal of process mining algorithms and the goal of our research (cf. Fig. 3.4). If we evaluate these examples based on the evaluation criteria in Section 8.4.1, process mining algorithms will have better scores in all behavior measurements. Consequently, we cannot generalize the evaluation results based on this single case. Instead, we consider the results as an interesting trigger for additional analyses on the behavior aspects of our mining algorithms.

Note that the scope of process mining is also broader. In this thesis we have assumed that process models are block-structured, activities have unique labels, and each process model can be built based on a limited set of process patterns. Process mining algorithms, on the contrary, can work without these constraints and consequently can be applied to more complex scenarios as well. Based on the analyses presented in this section, we consider our approach as an important complement of process mining algorithms. Instead of having one complex model which tries to cover all behaviors, we may maintain only a reference model of simpler structure, and allow process adaptations to obtain suited process variants supporting the execution of process instances best.

8.5 Summary

In this chapter, we have qualitatively and quantitatively compared our clustering and our heuristic algorithms for discovering a reference process model out of a collection of process variants. The clustering algorithm does not presume knowledge about the original reference process model based on which process variants are configured. By only looking at the variant collection, it can quickly discover a reference process model in polynomial time and provide additional information on how well each part of the discovered reference model fits to the variants. Our heuristic algorithm, in turn, can take the original reference model into account such that the user can control to what degree the discovered model differs from the original one. This way, we cannot only avoid spaghetti-like process models, but additionally control how much changes we want to perform. We further compared our techniques with existing process mining algorithms. Results indicate good performance of our algorithms in both structure and behavior aspect. Though the evaluation results are based on one example and we cannot generalize our conclusions, we consider them as an interesting trigger for integrating our approaches with process mining techniques.

9 Case Studies

In this section, we discuss how we applied our mining techniques to two real-world cases from the automotive and the healthcare domain. In particular, these two cases can be mapped to the two mining scenarios we have described in Chapter 1. To be more precise, the automotive case fits to Scenario 1, i.e., as input there exists a collection of process variant models, but no original reference process model is known. Opposed to this, the healthcare case can be mapped to Scenario 2, i.e., as in put there is a collection of process variant models as well as a documented reference process model to which the variants related. The healthcare and the automotive cases are described in Sections 9.1 and 9.2 respectively. We present a cross-case analysis in Section 9.3 and conclude this chapter with a summary in Section 9.4.

9.1 Hospital Case

9.1.1 Description

Context. We analyzed a variety of patient-centered processes in a large clinical centre (with more than 1000 beds) in Germany. In this clinical centre the diagnostic and therapeutic processes of a patient usually involve various, organizationally more or less separate units. For a patient being treated in a department of internal medicine or gynecology, for example, medical tests and procedures at the laboratory and the radiology department become necessary. In this context, medical procedures must be planned and prepared, and appointments be made. Further, specimen or the patient herself have to be transported, physicians from other units may need to come and see the patient, and medical reports have to be written, sent, and evaluated. Thus, the cooperation between organizational units as well as the medical personnel is a vital task, with repetitive but nevertheless non-trivial character. In order to optimize these processes and to better align them with the existing hospital information system, comprehensive reengineering efforts were conducted at the clinical centre resulting in several hundred process models. For capturing, analyzing and simulating these models two business process modeling tools were used - the ARIS toolset and the Bonpart tool (see Fig.

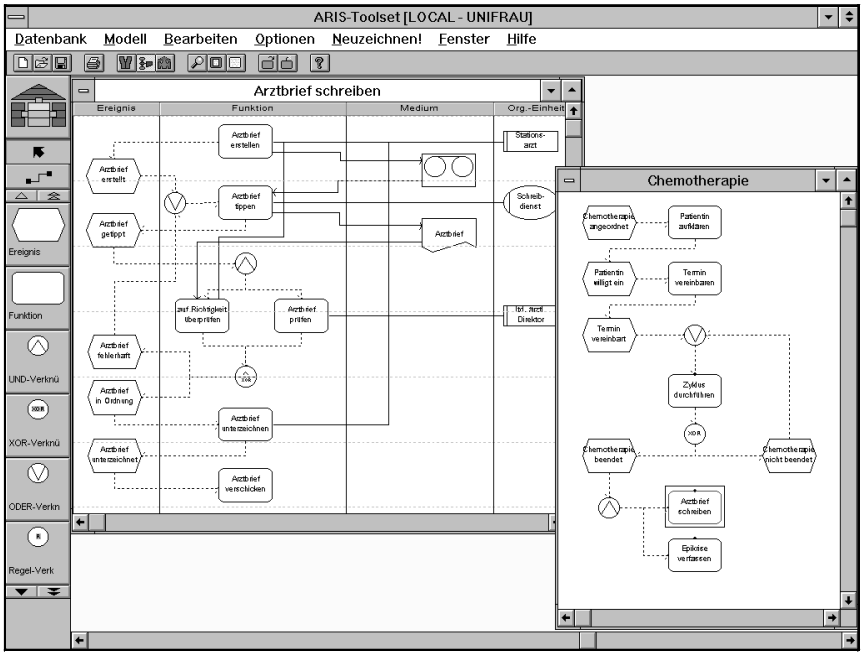


Figure 9.1: Two process models of a particular clinic as captured with the ARIS toolset

9.1 for a screenshot of one of the processes being modeled with ARIS toolset).

Data Source. We analyzed several process model repositories that emerged during the above mentioned process reengineering efforts. Particularly, we were able to identify more than 90 process variants for handling medical orders and medical procedures respectively (e.g., X-ray inspections, cardiological examinations, lab tests). Despite their similarity the different variants were captured in separate process models based on different notations (e.g., Event-driven Process Chains and Activity Diagrams) and modeling components (e.g., ARIS Architect, Bonapart). As example consider one variant of the medical order handling procedure as depicted in Fig. 9.2. The shown process variant was modeled with the ARIS toolset and captured as Event-driven Process Chain. Interestingly, all identified variant models were based on standard workflow patterns like Sequence, AND-/XOR-Splits, AND-/XOR-Joins, and Loop, and their size ranged from 7 to 17 activities. Interestingly, for each variant it was possible to transform its model into a behavior-equivalent, block-structured representation; i.e., it was possible to map the different variant models to a representation following our process meta model.[1] However, before this transformation, we had to apply a number of sim-

[1]This also applies to 70 other process models from the healthcare domain which we analyzed in another case study in close collaboration with a Women's Clinic. The respective models cover

ple refactorings (see [210] for respective techniques) to harmonize the identified process variant models. In particular, we had to relabel certain activities in order to obtain a consistent labelling style and to ensure that activities with the same meaning also have the same label. Besides this we excluded 3 process variants due to flaws in their corresponding model. Note that the notation we use in Fig. 9.3 is only for illustration purpose, but does not correspond to the original notation of the respective models (i.e., EPCs). Furthermore, we translated the German labels into English language to make the case better understandable for readers.

Sources of Variance. Though the variant models show structural similarities they comprise parts only relevant for a sub-collection of the variants. For example, some of the variants require confirmation of a medical order by a senior physician, while this is not required for other variants. Similarly, there exist medical procedures requiring complex scheduling activities, whereas in other cases no scheduling is required or the patient simply needs to be registered at the site of the care provider. Depending on the physical condition of the treated patient, in addition, a transport needs to be organized or not. Similarly, in emergency cases a short medical report is transmitted immediately after the medical examination to the requesting unit (e.g., a ward). Other variations of the analyzed models concern the preparation phases at the site of the wards and the examination units respectively.

Original reference model: The top of Fig. 9.3 shows the original reference process model S_{ref} as we could find it in the organizational handbook of the clinical centre. (Again not that the actual reference model was documented in terms of an Event-Process-Chain whose elements were labelled in German language.) Furthermore, this figure depicts four selected process variants as identified in our case study. In total, we consider the most relevant 84 process variants which make up more than 95% of the identified variants. Based on the number of corresponding process instances, we assign weights to the variant models ranging from 0.1% to 8.67%. However, none of the process variants is dominant, or significantly more relevant than others. Note that the original reference process model S_{ref} constitutes a very simple model since it only contains 7 activities organized in sequence. When discussing this reference model with process owners we learned that its original purpose was to define the basic organizational steps of an arbitrary medical order, and medical processes respectively but that the different variant models have evolved over time and thus more or less differ from this reference model.

We calculate average weighted distance between this original reference model and the given collection of process variants and obtain 5.307 as result; the corresponding fitness value is 0.585.

areas like birth and postnatal care, inpatient hemotherapy treatment, outpatient chemotherapy treatment, ovarian carcinoma surgery, and keyhole surgery.

[2]Note that the labels of the depicted Event Process Chain are in German since the respective project was conducted in a German clinical centre.

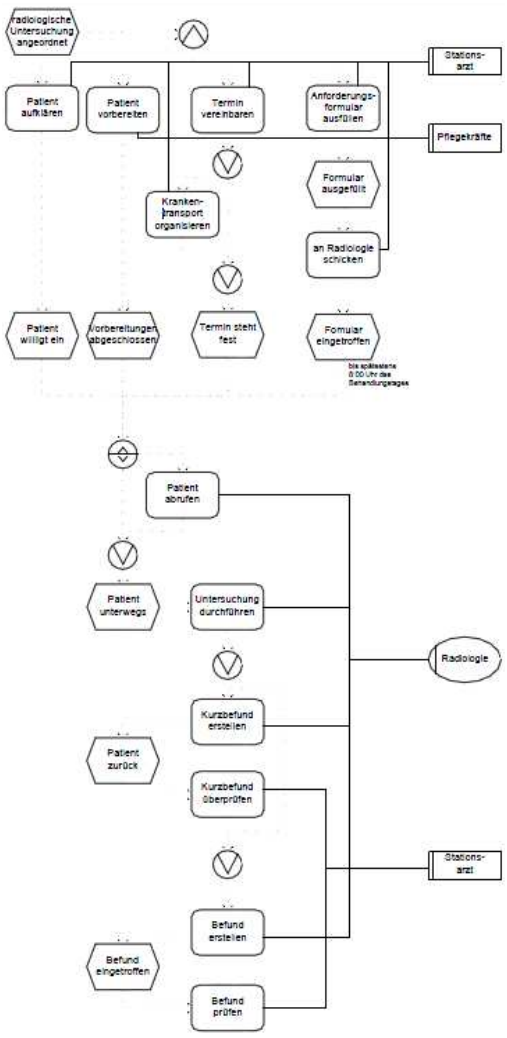


Figure 9.2: Example of one variant of the medical order handling process[2]

9.1.2 Results

We applied our heuristic algorithm for variant mining to this hospital case. We did not set limitations regarding the number of search steps, i.e., the algorithm continued as long as it was able to discover a better models. Table 9.1 shows obtained mining results and measures in respect to the original reference model, the intermediate process models, and the finally discovered reference process model. Comparable to the analysis of our illustrating example from Table 7.2, we calculate fitness value and average weighted distance for every discovered model. We

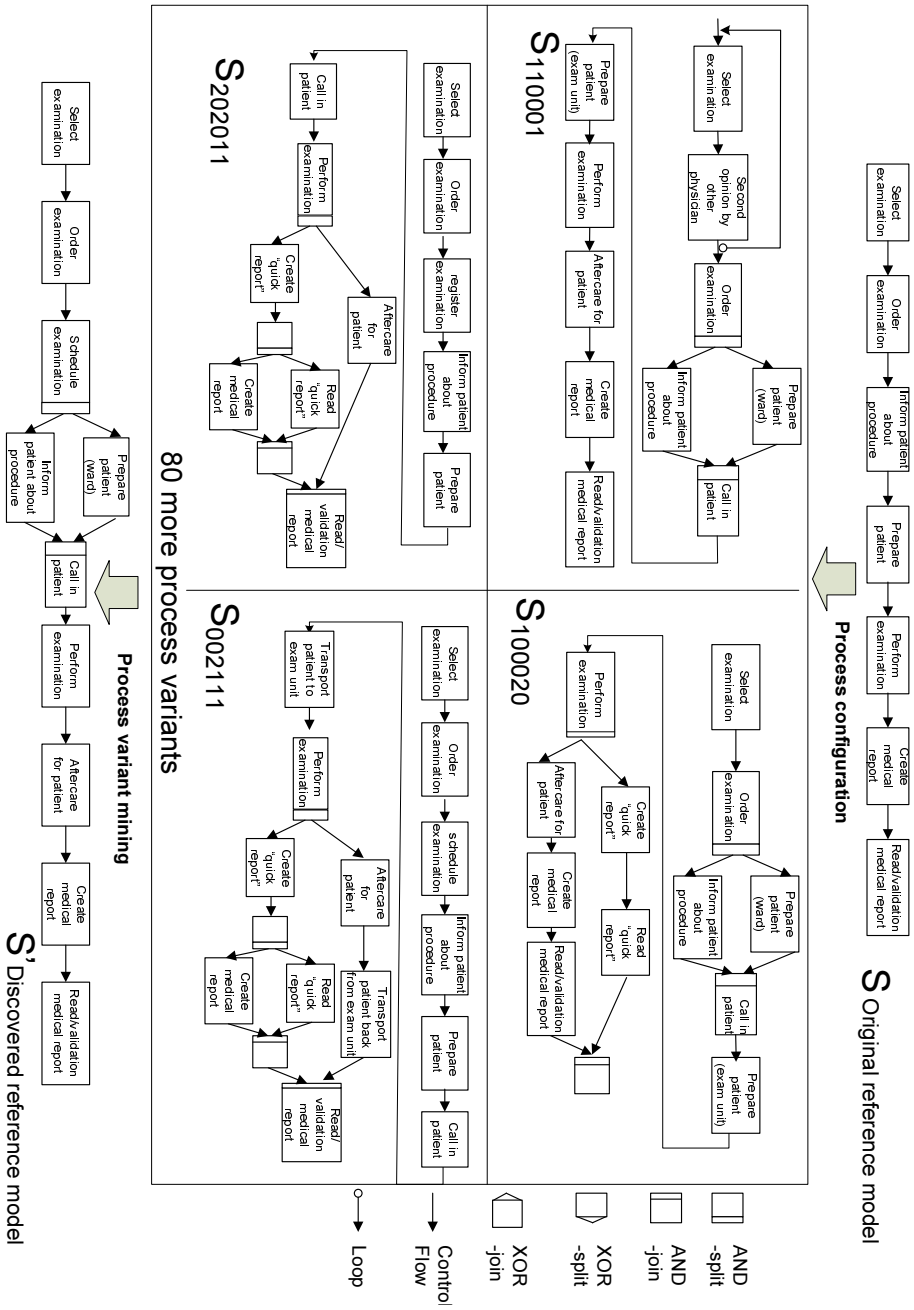


Figure 9.3: Selected process variants from the healthcare case

	S	R_1	R_2	R_3	R_4
Model name	Original reference model	1st intermediate model	2nd intermediate model	3rd intermediate model	Final reference model
Fitness	0.585	0.673	0.759	0.773	0.778
Average weighted distance	5.307	4.307	3.401	2.981	2.795
Change type		Insertion	Insertion	Insertion	Move
Delta-fitness		0.087	0.086	0.014	0.005
Delta-Distance		1	0.905	0.421	0.186

Table 9.1: Mining results of the hospital case study

further document delta-fitness and delta-distance (cf. Chapter 7) value for every change operation. Results can be summarized as follows:

1. *Time.* The time to perform the mining is neglectable, it takes only 0.782 second to discover the three intermediate models R_1, R_2, R_3, and the final result R_4.
2. *Improvement of average weighted distance.* If we do not set any limitation on the number of search steps, we are able to reduce average weighted distance of the reference process model from 5.307 to 2.795, which corresponds to 47.3% less than average weighted distance between original reference model and variants. This result again indicates that we are able to significantly reduce future configuration efforts when applying our algorithm.
3. *Precision.* Precision of the results is 100% for this case. We are able to discover a better process model with lower average weighted distance to all 4 change operations.
4. *Importance of top changes.* Delta-distance and delta-fitness are monotonically decreasing in this case, indicating that a change operation performed before another one is always more important than this other one. In this case, the top 50% change operations account for 75.83% of the overall distance reduction.

It is not difficult to conclude that our algorithm performs well for this hospital case. We are able to discover a better process model in a very short time, and the models we discover also satisfy the features of our algorithms, i.e., our mining algorithm discovers more important operations at the beginning of the search, while it considers the less important one at th end.

We have shown the newly discovered reference model (as depicted at the bottom of Fig. 9.3), to process owners at the clinical centre who confirmed that this new reference process model is closer to the variants than the old reference model. We additionally considered other process models which were related and did show some variance; e.g., we analyzed process variants representing different kinds of chemotherapeutic treatments. However, since we were only able to identify three different variant models of this process, we omit further details here.

The clinical centre was particularly interested in the harmonized variant models of the medical order handling procedure and the optimized reference process

model we derived for it. Respective artifacts were needed for discussing required customizations of the hospital information system in-use.

9.2 Automotive Case

9.2.1 Description

Context. We conducted a case study in a large automotive company in which we analyzed variants of its product change management process. Basically, this process comprises several phases like specification of a change request, handling of this change request, change implementation, and roll-out. In the following we only consider the top-level process and comment on sub-processes later on. Usually, the change management process starts with the initiation of a Change Request (CR), which must then be detailed and assessed by different teams (e.g., from engineering and production planning). The gathered comments have to be aggregated and approved by the CR board. In case of positive approval change implementation may start (e.g., detailing the planning and triggering the re-engineering of parts affected by the change).

Data Source. We identified 14 process variants dealing with (product) change management. These variants were captured in separate process models being expressed in terms of UML Activity Diagrams and using standard process patterns like Sequence, AND-/XOR-Splits, AND-/XOR-Joins, and Loop. Size of the variants ranged from 5 to 12 activities and their weights from 2 to 15 according to the relative frequency of corresponding process instances. However, none of the process variants was dominant or significantly more relevant than the others. All variant models were already block-structured or could be easily transformed into a behavior-equivalent block-structured process model. Note that the models depicted in Fig. 9.4 resulted from a transformation to the specific graphical notion as used in the context of this thesis. The actual variant models have been expressed in terms of UML Activity Diagrams.

Sources of Variance. Though the variant models show structural similarities they comprise parts which are only relevant for a sub-collection of the variants. For critical changes, for example, the Quality Assurance Department needs to be involved in the appraisal and commenting of the change request, while this is not required for normal changes. Concerning low-cost changes, in turn, change implementation may start before the change request is finally approved. In this case, the implementation procedure will have to be aborted and compensated if the the approval is withhold. Other points of variations concern the preparation of the approval task, the communication of implemented changes, and the triggering of secondary changes (raised by the requested one). The left hand side of Fig. 9.4 shows 4 sample variant models from our case.

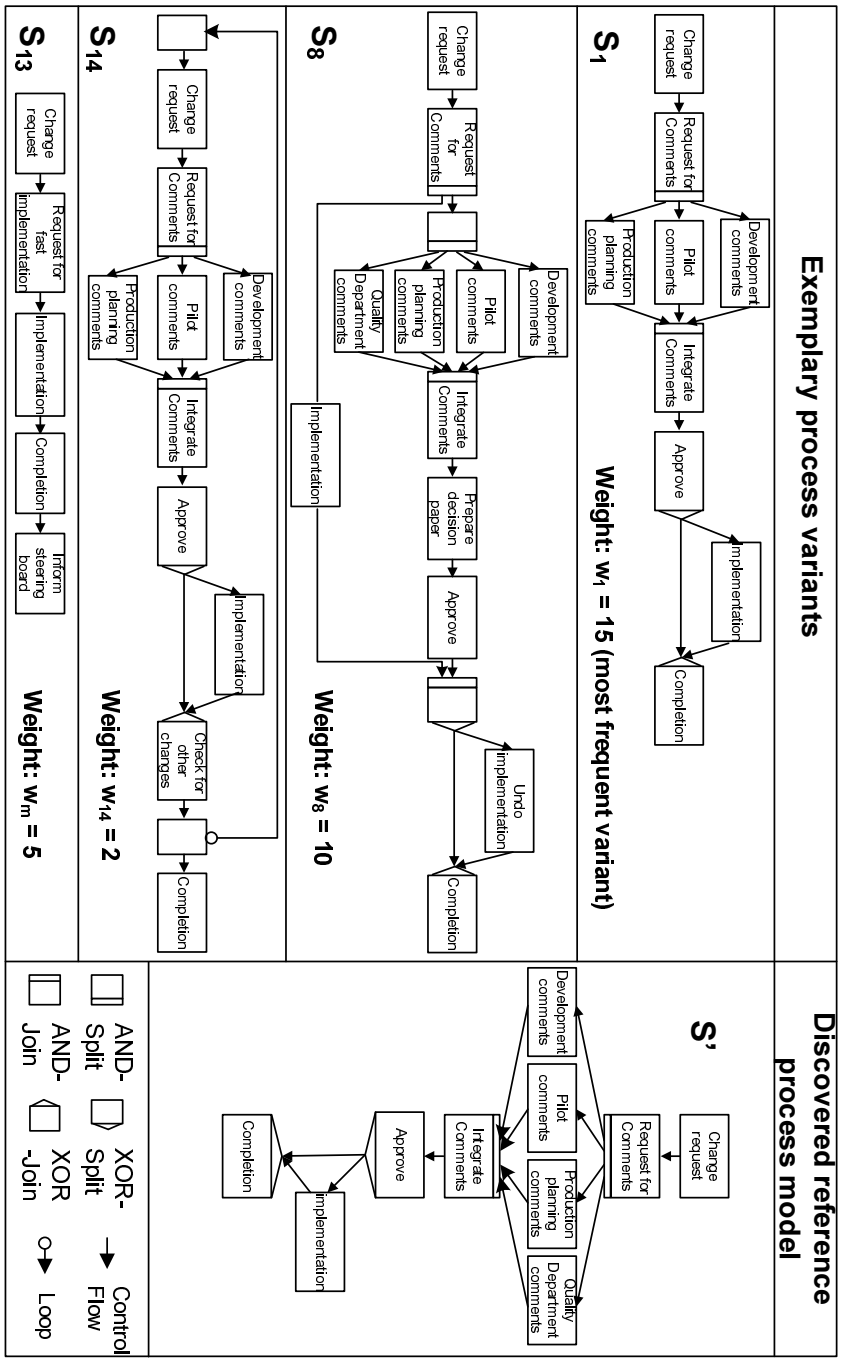


Figure 9.4: Selected process variants from the change management case

9.2.2 Results

Since we did not know the original reference process model, this case can be mapped to Scenario 1. Therefore, we applied our clustering algorithm to "merge" the process variants. This way we obtained S' (cf. Fig. 9.4) as reference process model. As average weighted distance between S' and the variants we obtained 2.06. The time to find the model was negligible (0.031 seconds).

We discussed the discovered reference model with process engineers from the automotive company. They confirmed that it constitutes a good choice for representing the top level change management process.

Based on the discovered reference process model (or base model for configuration), we can apply advanced change management techniques to configure this reference process model into the different process variants in an effective and manageable way. In the automotive company, in which we conducted the case study, the Provop research project was launched in which advanced concepts for the management and configuration of process variants have been developed [68]. In Provop a particular process variant can be configured by domain experts by adjusting a given reference process model through applying a set of high-level and predefined changes.

In particular, the mining algorithms we developed in this thesis can significantly speed up the design of such a reference process model; more precisely respective reference models can be automatically discovered for any collection of block-structured process variant models. When further applying our mining algorithms to sub-processes relating to the different phases of the change management process (e.g., change implementation) and their variants we obtained good results as well.

The discussed case constitutes one of many process scenarios we encountered and analyzed in the automotive domain. Interestingly, for almost all of them we were able to identify large collections of similar process variants, each of them being valid in a particular application context. Regarding the presented case the process owners liked the discovered reference process model and considered it as very intuitive. Based on this result, they asked us to apply our mining approach to the more specific phases of the change management process as well, which resulted in well-accepted reference models for its sub-processes as well. We also studied other sources of data. Regarding release management for electric/ electronic components in a car, for example, we identified more than 20 process variants depending on the product series, involved suppliers, or considered development phases.

Another complex scenario we considered was the product creation process, for which dozens of variants exist. Thereby, each variant is assigned to a particular product type (e.g., car, truck, or bus) with different organizational responsibilities and strategic goals, or varying in some other aspects. Regarding the latter case, however, we encountered additional problems concerning the inconsistent labeling of activities, the use of different process granularities, and the heterogeneity of the used modeling formalisms. This also shows that our algorithms need to be

integrated in a larger process repository framework, which additionally provides support for model configuration, model refactoring, and model management [210, 68].

9.3 Cross-Case Analysis

Our case studies fit to the two scenarios we described in Chapter 1. Therefore we can apply the results from Section 8.3.1 to qualitatively compare the two cases (cf. Table 8.1). For example, the two cases have different input data. While our automotive case does not comprise a reference process model, our healthcare case does. The cases also have different result formats: we discovered a reference process model in our automotive case, while identifying a sequence of change operations in the context of the healthcare case. Finally, in the context of the two cases, we applied different evaluation criteria (separation and cohesion in the context of the automotive case, and the described fitness function in connection with the healthcare case).

As discussed in Chapter 8, our clustering and heuristics algorithms do not exclude each other. In principle, we can apply the heuristic algorithm to the automotive case (no reference model) and the clustering algorithm to the healthcare case as well. By doing this, we can quantitatively compare our cases.

Since there was no original reference process model in the automotive case, we used the most frequent variant (cf. S_1 in Fig. 9.4) as starting point of our search. We did not set any search limitation in this context such that our heuristic algorithm could discover the best model. As result we obtained process model S' (cf. Fig. 9.4) as best reference process model after performing one change on S_1, which is the same model as we discovered using the clustering algorithm. Though the heuristic algorithm ran longer than the clustering one to find the reference process model, overall search time was only 1.062 seconds.

For the healthcare case, we can apply our clustering algorithm purely based on the collection of variants. By setting the threshold to 50%, we obtain a process model S'' which is similar to S' from Fig. 9.3. S'' contains one additional activity `prepare patient (exam unit)` as successor of `call in patient` and predecessor of `perform examination`. Though different, S'' has the same average weighted distance to the variants as S' has, and only takes 0.031 seconds to discover the result. However, since we did not consider the original reference process model S, it is not possible to observe how the reference process model evolve as our heuristic algorithm can do (cf. Table 9.1).

9.4 Summary

This chapter described two applications of our algorithms to real-world cases from two different domains. The automotive case solely comprises a collection of process variants. Therefore, we applied the clustering algorithm to discover a

reference model which can be easily configured into these process variants. We discussed the discovered reference model with process engineers from the automotive company and they confirmed that it constitutes a good choice for representing the top level change management process. Furthermore, an additional research project was launched by the respective automotive vendor in which a component for the management and configuration of respective process variants was implemented. In particular, the techniques provided by this thesis are complementary to this component and enable the design of optimized process reference models. Regarding the healthcare case, we were confronted with both a collection of process variants and a reference process model out of which these variants were configured. We then applied our heuristic algorithm to improve the existing reference process model. We presented the newly discovered reference process model to process owners at the clinical centre who confirmed that it is closer to the variants than the old reference model. The identified reference model was further used as artifact in discussions with the vendor of the respective hospital information system in order to facilitate required customizations and configuration options.

Altogether, from these two and other cases we considered, we believe that the practical relevance of our algorithms will be high.

Part IV

Conclusion

10 Related Work

Though clustering and heuristic search algorithms have been widely used in areas like data mining [181], artificial intelligence [110] and machine learning [138], respective techniques have not been applied in the context of process variant mining so far; i.e., there exist no advanced approaches for learning from the adaptations that were applied when configuring process variants out of a given reference process model.

In this chapter, we discuss the body of knowledge with respect to process variant management and process variant mining (cf. Fig. 10.1). As the most relevant research field for our work, we first extend our discussions on process analysis and process mining algorithms in Section 10.1. Section 10.2 then presents related work on traditional data analysis and data mining techniques. As an emerging new field, Section 10.3 introduces approaches on web service support. Following this, we look at general evaluation methods for algorithms in Section 10.4. Finally, Section 10.5 concludes with a summary.

10.1 Process Analysis and Process Mining Approaches

In this section, we discuss related work from the process management area. Particularly, we focus on related approaches on process change (cf. Section 10.1.1), process similarity (cf. Section 10.1.2), process mining (cf. Section 10.1.3), conformance checking (cf. Section 10.1.4), process change mining (cf. Section 10.1.5), and reference modeling (cf. Section 10.1.6).

10.1.1 Process Changes and Process Variant Management

The ability to effectively deal with process changes has been identified as one of the most fundamental success factors in process-aware information systems (PAISs) [124, 133, 149, 211, 213]. Considerable efforts have been made to make PAISs more flexible [217, 66, 141, 165, 50, 189, 133]. For example, late binding [1] and late modeling [109] techniques defer modeling decisions to run-time by enabling users to select process fragments and to specify the control dependencies between them on-the-fly. Declarative approaches [189, 133, 214] further enhance

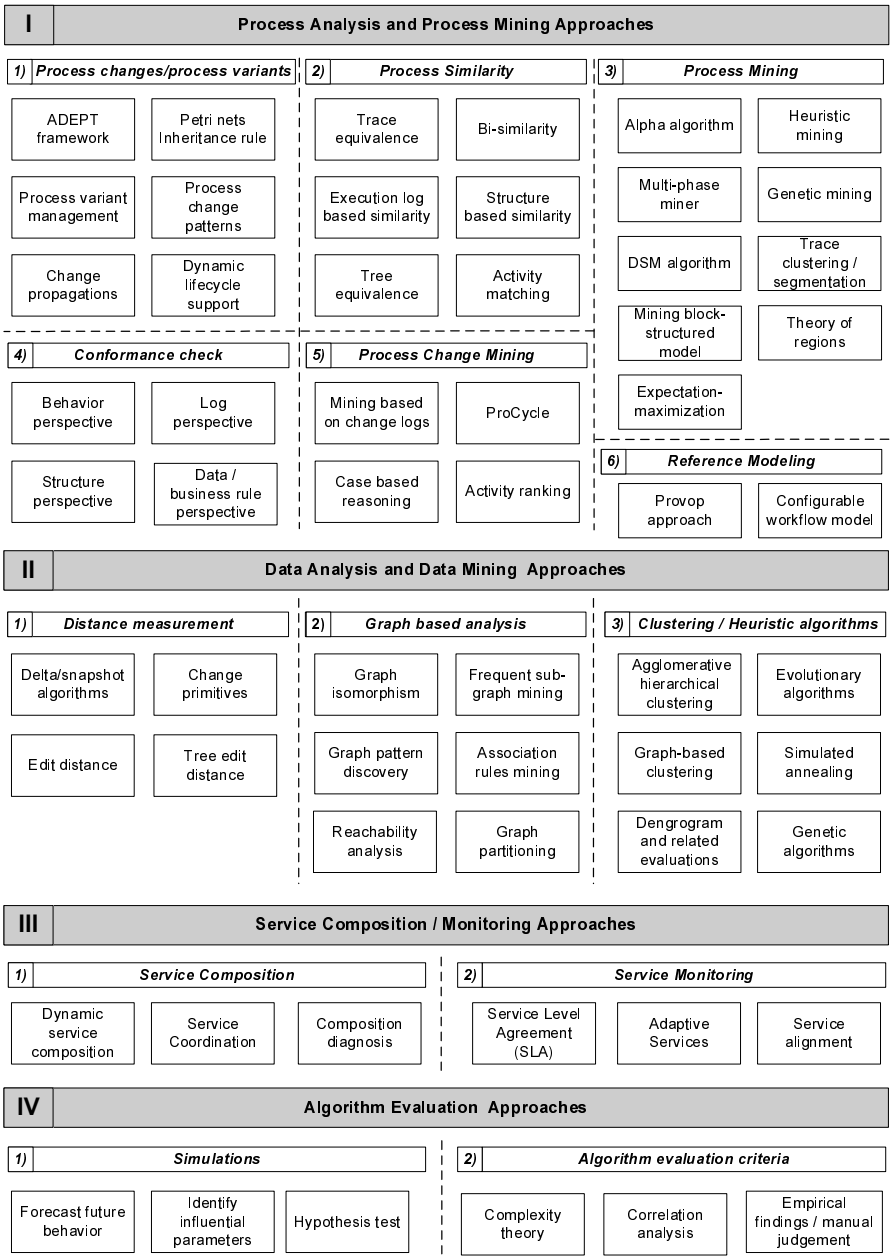


Figure 10.1: Overview of related work

flexibility by only providing a set of rules and constraints, so that users can compose a process model flexibly. Based on inductive logic programming, [50] allows for flexible process execution and planning on condition that certain rules expressed in terms of predicate logic are satisfied.

Furthermore, structural process changes at runtime and approaches for flexible process configuration have been intensively discussed in the literature [154, 155, 211] (cf. Section 2.2 for a detailed introduction). In this context, change patterns and process flexibility frameworks have been introduced to describe common change features and flexibility approaches in the field of process management [215, 211, 189, 133, 50]. A comprehensive analysis of theoretical and practical issues related to (dynamic) process changes, for example, has been provided in the context of the ADEPT2 change framework [141, 148] (cf. Section 2.3 for details). Based on its conceptual framework, the AristaFlow BPM Suite has emerged [148, 145, 37]. AristaFlow BPM Suite is an industrial-strength version of the ADEPT2 process management system and has been already successfully applied in various application domains [94].

There exist approaches for dynamic structural changes of Petri nets as well. In this context, inheritance rules were introduced which allow modifying a Petri Net, while maintaining its soundness [187]. Based on the suggested theory, tools like C-YAWL and C-EPC [162, 54] for configurable process models [55] have emerged which allow configuring process models at both runtime and buildtime. For example, users may activate, block or hide certain activities or process fragments of the given reference process model. However, when configuring a process model, certain requirements need to be fulfilled in order to ensure that the resulting process model meets correctness constraints [184, 183]. A similar approach is provided by Provop [66, 68], which allows for the configuration of process variants by applying well-defined change patterns (e.g., insert, delete, and move process fragments) to a given reference process model. Thereby, Provop assists designers in the context-based application of respective changes when configuring a a process reference model to a particular context [65], Provop further ensures correctness of the configurable process variants in this context [67].

Change propagation deals with the problem that changing one process model may influence dependable ones; i.e., a change applied to one process model needs to be propagated to relating process instances and their models [156, 220, 229]. In this context, approaches like ADEPT2, WIDE and WASA2 [147, 27, 156, 223] provide solutions to propagate process schema changes to related process instances while further guaranteeing the correct executability of these process instances. [229, 159] allow change propagations between process models of different abstraction levels based on their behavioral profiles. It discusses the influence of a process change on related partner processes, and provides an approach to propagate such change to partner process models based on process choreography analysis.

Furthermore, many efforts have been undertaken to enable PAISs to provide full process lifecycle support. In this context approaches like ProCycle [213, 158], CAKE2 [119], WASA2 [224, 207], TRAMs [87], Worklets/Exlets [1, 2], and YAWL [192] have emerged (for an overview see [154, 211]). They are all trying to provide

users with the flexibility to dynamically adapt the processes running in the PAIS to real-world situations at both process type level and process instance level.

Finally, there exist approaches which provide support for the management and retrieval of separately modeled process variants. As example, [107, 108] allows for storing, managing and querying large collections of process variants within a process repository. Graph-based search techniques are used in order to retrieve variants that are similar to a user-defined process fragment. Obviously, this approach requires profound knowledge about the structure of stored processes, an assumption which does usually not hold in practice. However, no techniques for analyzing the different variants and for learning from their specific customizations are provided.

10.1.2 Process Similarity

Various papers have studied the process similarity problem [188, 187, 231, 12, 73]. Trace equivalence is commonly applied in this context to decide whether two process models are similar or identical [73]. More precisely, two process models are considered being the same if they can generate the same trace set (cf. Def. 4 in Section 2.1). Bisimulation [187, 203] refines the notion of trace equivalence by considering stronger notions from structural aspects, e.g., the branching time of different models. Usually respective similarity measurements result in binary "Yes" or "No" answers. Consequently, their usefulness is limited in our context.

Using traces, [188] measures the similarity between two process models based on the observed behavior as captured in execution logs. This way, the similarity also reflects the relative importance of each single trace from the execution log. The approach presented in [231], in turn, uses edit distance to measure the difference between the trace sets of two process models. More precisely, the distance between process models is measured in terms of the sum of all edit distances between traces from the two trace sets. Other similarity measures use *precision* and *recall* to evaluate the differences between two process models [188, 134]. Finally, [202] provides an interesting approach to measure the similarity between two process models by using causal footprints, which describe a collection of the essential behavioral constraints imposed by a process model. This approach takes both behavior and semantics into account, and provides a similarity score close to manual judgments.

There are few techniques measuring similarity between process models based on their structure. Regarding Petri Nets and state automata, similarity between process models can be measured based on behavioral inheritance rules [211, 191, 187]. For example, [191] introduces the notion of greatest common divisor (GCD) and least common multiple (LCM) of two Petri Nets. In this context, the GCD of two Petri Nets refers to a Petri Net which captures the maximal commonality between them; the LCM of two Petri Nets, in turn, is a Petri Net which captures each conceivable behavior of the two Petri Nets with simplest structure. Finally, [46] presents an approach for measuring similarity between BPEL process models taking into account their common structure. Such technique is similar to our

distance measurement (cf. Chapter 5) but can only provide a binary answer to whether two models are the same or whether one model is a sub-model of the other. We refer to [219] for an overview of approaches measuring behavioral and structural similarities between process models.

Similarity measurements have been also applied for matching activities in process models [44, 43, 24]. In this context, the goal is to map the activities from one process model to the activities of another one, while taking semantics, structure and control flow aspects into account. Similarity results are considered being helpful when merging two process models or when retrieving them from a process repository [43].

None of these approaches measures similarity in terms of a unique number, based on the effort needed for transforming one process model into the other as presented in this thesis.

10.1.3 Process Mining

Process mining has been extensively studied in literature. Its key idea is to discover a process model by analyzing the execution behavior of (completed) process instances as typically captured in execution logs [195] (cf. Section 2.5 for a detailed introduction of process mining techniques). In the context of process mining, a variety of techniques has been suggested with different properties and goals [195, 221, 39, 197], and advanced tools like ProM [201, 64] have been developed supporting a large spectrum of process mining and process analysis algorithms. For example, the Alpha algorithm [197, 222] can quickly discover a process model expressed in terms of a Petri Net [126]. However, this simple algorithm is only of theoretical interest since it cannot handle short loops or noise. It can also not differentiate between important traces and trivial ones. The heuristic mining algorithm [221] extends the Alpha algorithm by additionally being able to cope with noise in the logs. In this context, noise often refers to exceptional behavior or erroneous data contained in the execution log. The genetic mining algorithm [39] additionally extends process mining techniques by also considering complex constructs like non-free choice, non-unique labels and short loops. When applying the genetic mining algorithm, the discovered process model will be closer to the behavior as observed from the execution log. However, this algorithm is also more complex than the Alpha or the heuristic mining algorithm.

Besides the aforementioned algorithms, which discover a Petri Net from an execution log, several other process mining algorithms exist which discover a process model being expressed in terms of other formalisms. For example, [33] introduces three algorithms representing different levels of accuracy and enabling different degrees of robustness at the presence of noise. Regarding these algorithms, traces are first converted into an event graph based on Markov chains [10, 72] and are then transformed into a finite state machine which represents the discovered model. Multi-phase Miner [200] discovers an EPC model by iteratively including each trace in the discovered process model. The approach presented in [172], in turn, focuses on the discovery of a block-structured process

model. This algorithm iteratively refines the internal structure of a bigger block into several smaller ones. This iterative approach continues until all blocks are identified. Using the Expectation-Maximization algorithm, [48] tries to discover a process model from unlabeled execution logs, i.e., without having information about which activities belong to which process instances.

In order to achieve better results in process mining, trace clustering techniques have been suggested [19, 57, 177]. Instead of considering the whole trace set for process mining, traces are first clustered into several smaller sets, and then a collection of process models is discovered for each of these small trace sets using standard process mining techniques like the aforementioned heuristic and genetic algorithms [39, 221]. Consequently, instead of one several process models are discovered simultaneously. Each of them fits better to the corresponding subset of traces when compared to the process model that can be discovered when considering all traces. However, the discovered collection of models still needs to be managed separately. So far, it has been not possible to merge them into a generic model from which they can be easily configured.

Another direction of trace clustering aims at clustering activities within traces for better visualization purpose [63, 206, 15]. For example, [63] clusters several activities into a segment and then discovers process models only based on these process segments. This way spaghetti-like structures are avoided for the discovered model. Similarly, the approaches presented in [206, 49] apply techniques based on Markov chains [72, 10] in order to divide a long trace into a number of small clusters. Based on these trace clusters, several process fragments, which describe different parts of a process model, can be discovered independently.

Based on the theory of regions [35], we can create a process model by adding all process variants to different branches of an XOR-split. Then we can simplify this process model and consequently obtain a model which captures all behaviors the process variants can show. Such state-based mining techniques inspired Multi-phase Miner [200] or Region-based Mining [199] in the field of process mining. For example, Multi-phase Miner follows an iterative approach, and in every iteration it improves the discovered process model by including one more trace in the model. Consequently a key property of Multi-phase Minder is that it can always discover a process model with a fitness value of 1, i.e., all observed behaviors in the trace set are covered in the discovered process model. However, the process models discovered from these approaches often result in over-fitting process models with a spaghetti-like structure (cf. Chapter 8 for detailed explanation). Usually, state-based algorithms also have difficulties to cope with noise or to deal with infrequent process variants.

As we have already systematically discussed in Chapters 3 and 8, traditional process mining differs from process variant mining due to its different goals and input data [99]. The goal for mining process variants is to discover a reference process model based on which the variants can be configured with less efforts. Opposed to this, the goal of process mining is to discover a process model which covers the observed behavior as it is captured in execution logs best. In principle, we can apply traditional process mining algorithms to the problem addressed

by this thesis as well. However when conducting a quantitative comparison, we learned that the clustering and the heuristic algorithms presented in this thesis can discover a process model with better structure, but without sacrificing the behavior aspects too much (cf. Chapter 8 for details). [79] presents a method to mine configurable process models based on event logs, but is still focusing on the discovery of process models from event logs rather than on the reduction of efforts for structural process configurations.

10.1.4 Conformance & Compliance Checking

Conformance checking techniques have been widely used to measure the matching between designed process model and its actual executions [168]. Such analyses often can be interpreted from both log perspective and process model perspective. From the process model perspective, conformance checking measures to what degree process executions deviate from the originally designed process model; by contrast the log perspective reflects to what degree the real behavior is covered by the original process model [168]. Such idea has also influenced process mining approaches like genetic mining [39] and trace clustering [19]. Similarly, conformance checking can be applied in the context of process monitoring where people focus on monitoring how business process executions deviate from the original process model [58, 237].

In addition, conformance checking can be used to evaluate a process model from other perspectives as well. For example, behavioral appropriateness measures to what degree extra behavior, which is not captured in the log, is included by a process model [168]. If a process model contains too much extra behavior, it is considered as "over-fitting". Finally, conformance checking can also be used to evaluate the complexity of a process model [168].

Similarly, compliance checking techniques measure to what degree data-flow or business rules are satisfied by a process model [5]. Respective techniques are often applied when designing a new process model or when auditing past executions of existing business process models [52]. However, conformance or compliance checking techniques are often focusing on the execution, the data flow or related business rules of a process model, but not on structural adaptations as addressed in this thesis.

10.1.5 Process Change Mining

There exist few techniques for mining process model variants. For example, the ProCycle system enables change reuse at the process instance level to effectively deal with recurrent problem situations [158, 213]. ProCycle applies case-based reasoning techniques to allow for the semantic annotation as well as for the retrieval of process changes. Based on this, respective process adaptations can be re-applied in similar problem context to configure other process instances later on. If the reuse of a particular change exceeds a certain threshold, it becomes a candidate for adapting the process schema at the type level; i.e., for evolving

this schema accordingly and thus for considering the change for future process instances as well. Though the basic goal of ProCycle is similar to our approach, the techniques applied are much more simpler and do not consider variation in changes. Instead, the focus of ProCycle is more on the reuse of ad-hoc changes based on semantic annotations. A similar learning approach, which also relies on case-based reasoning techniques, is provided by CAKE2 [119, 216].

To mine high level change operations, [61, 62] present an approach based on process mining techniques. The input consists of a change log, which explicitly documents all process changes. Process mining algorithms are then applied to discover the execution sequences of the changes (i.e., the change process). However, a prerequisite of this approach is the presence of a valid change log which is not always available in practice. In addition, this approach simply considers each change as individual operation such that its result is more like a visualization of changes rather than their mining. By contrast, the clustering and heuristic algorithms presented in this thesis can discover a reference process model without requiring the presence of a change log.

The work presented in [104] introduces a technique to rank activities based on their potential involvement in process configurations. Similar to our heuristic algorithm, it can identify which activities are more often changed than others. However, opposed to our heuristic algorithm it cannot provide suggestions on how to change these activities in order to improve the reference process model [105].

10.1.6 Reference Modeling

Regarding Configurable Workflow Models [55], all process variants are merged into one core reference process model. This merging is based on inheritance rules known from Petri Nets [187]. Though techniques like questionnaire-based configuration can help in making the right configuration decisions at configuration time [163], the resulting model turns out to be complex containing plenty of decision points (see the case study reported in [56]). This approach becomes even more difficult when being confronted with a large collection of process variants, not being equally important. In this case, an extremely complex process model might result which contains too many decision points and which cannot differentiate between important variants and trivial ones. In fields like healthcare, such complex models are often not preferred due to the large efforts needed to use them [4, 96].

In this context, the aforementioned Provop approach (cf. Section 10.1.1) provides more flexibility in defining a reference process model and its configuration options [68]. However, it still requires manual discovery of a reference model (or a based model for process configurations). When encountering a large collection of complex process variants, this may require considerable time and efforts to discover a good reference model. The algorithms presented in this thesis can act as a decision support tool for discovering a reference process model in the context of Provop approach.

10.2 Data Analysis and Data Mining Approaches

We first introduce and compare different distance measurements in Section 10.2.1. Following this we discuss techniques for graph-based analyses in Section 10.2.2. Section 10.2.3 then provides a general discussion of clustering and heuristic algorithms.

10.2.1 Distance Measurement

In the database field, the delta-algorithm [93, 31] has been widely used to measure the difference (i.e., delta) between two datasets. When updating a database, only the delta is performed in order to minimize the number of transactions of this update. Similar to the design goals of this thesis, the idea of minimizing the number of changes is commonly applied in fields like data warehousing [93] and distributed systems [31]. In addition, such techniques have been applied for measuring process model difference. For example, the approach presented in [7] measures the distance between two process models by determining their difference in respect to their node and edge sets.

In the context of text processing, the edit distance (or Levenshtein distance) is used to measure the minimal number of changes required to convert one string into another [231, 208, 166]. Such distance measurement is commonly used for string and text processing [9, 208]. It has been also applied for various analyses of the traces a process model can produce, e.g., for measuring process distance [231] and for trace clustering [177, 19].

Tree edit distance extends edit distance by analyzing tree structures instead of strings [12]. Due to is ability to handle more complex structures, this technique is frequently applied in fields like computational biology [80] and compiler optimization [180].

However, it would be not a good idea to directly apply the above mentioned approaches in our research context. In particular, they do not take the structural aspects of process models into account. A process model contains richer information than just nodes and edges (e.g., concerning split and join semantics), and various properties (e.g., soundness) need to be guaranteed when changing it [225, 211].

10.2.2 Graph-based Analysis and Mining Approaches

A process model is often represented as a graph structure based on which different kinds of analyses are performed [7, 108, 188]. Informally, a graph consists of a set of nodes, which can be connected using (directed) edges. Graph-based data representation is used in many research fields like computer networks, social networks and bioinformatics. Consequently, we can utilize some of the results from graph theory in the context of mining and analyzing process models as well.

Graph isomorphism [181] and sub-graph isomorphism [181, 86, 235] are used to measure the similarity between two graphs. In this context, two unlabeled graphs

(or sub-graphs) are compared based on their structure, and they are considered being isomorphic if a one-to-one mapping between their nodes exists. Despite the high complexity of this problem, most algorithms can only provide a binary answer, and are therefore difficult to apply to the basic problems discussed in this thesis.

There are few techniques which allow to learn from process variants by mining recorded change primitives (e.g., to add or delete control edges). For example, the approach presented in [7] measures process model similarity based on the adjacency matrices of the process models and suggests mining techniques using this measure. Similar techniques for mining change primitives exist in the field of association rule mining [181, 228], frequent sub-graph mining [78, 91], and graph pattern discovery [235]; here common edges between different nodes are discovered in order to construct a common sub-graph from a set of graphs. Such techniques have been commonly applied in the field of bioinformatics for "subdue" discovery, where "subdue" represents a certain sub-structure of genes or proteins, which has a particular chemical or biological behavior [75]. Furthermore, social network researchers use these techniques to find communities in which people communicate internally and share common interests [121]. We refer to [29] for a survey on graph mining topics and algorithms.

Besides mining, various analyses are enabled using graph representations. For example, the shortest path algorithm [34] can be applied to find the shortest path from one node to another; reachability analysis [166] can be applied to figure out whether there are graph nodes that are isolated from other nodes. Based on advanced techniques, we can identify key players in a network as well [18]. Key players are the ones whose removal would disrupt or disconnect the network maximally. In addition, we can evaluate the performance of networks, or analyze a particular network phenomenon using simulations. For example, [25] simulates virus propagation in order to stop the infection before it becomes an epidemic.

All these approaches are based on graph representations. However, they do not consider important properties of process meta models. For example, they do not consider soundness issues, do not differentiate between AND-Split and XOR-Split nodes, and have difficulties to cope with silent activities (i.e., unlabeled process activities without any associated action).

10.2.3 Clustering and Heuristic Algorithms in General

The clustering algorithm introduced in this thesis has been inspired by agglomerative hierarchical clustering and can be considered as a variant of it [181]. When compared to traditional agglomerative hierarchical clustering approaches, our clustering algorithm differs in technical details, e.g.; in how to build a block after finding a cluster and in how to reset the datasets afterwards (see Chapter 6 for details). Various kinds of graph analyses also employ clustering techniques; e.g., for partitioning a graph or for finding its minimal spanning tree [82, 181]. These approaches have been applied in various domains as well; e.g., to speed up text processing [42] or to optimize the resource allocation in a distributed system

[111]. In addition, the evaluation measures we apply in the context of our clustering algorithm - precision and separation - were inspired by dengrogram analysis for clustering algorithms [181]. In most clustering algorithms, precision evaluates to what degree data samples are clustered into the right clusters, while separation measures to what degree different clusters are separable from each other [181]. In the context of this thesis, we use similar computation approaches with minor adaptations.

Heuristic algorithms or metaheuristic approaches have been applied in various fields, including data mining [181], artificial intelligence [110] and machine learning [138]. Normally, a heuristic algorithm optimizes a problem by iteratively trying to improve a candidate solution (a search method) with regards to a given measure of quality (fitness). A problem employs heuristics when "it may have an exact solution, but the computational cost of finding it may be prohibitive" [110]. Although heuristic algorithms do not aim at finding the "real optimum", they are widely used in practice. Particularly, heuristic algorithms have been widely applied in solving problems with $\mathcal{NP}$ complexity, like traveling salesman, integer linear programming, and resource-constrained project scheduling problems [97, 106, 34]. The heuristic algorithm introduced in this thesis (cf. Chapter 7) constitutes an evolutionary algorithm (or hill-climbing approach) [110]; i.e., we try to iteratively find better solutions. As extension, genetic algorithms further imitate biological evolutions by breeding generations of offsprings to find the offspring with the best gene [176]. Simulated annealing further extends evolutionary algorithms by continuing search with less optimized solutions so that it can avoid trapping into a local optimum [85]. In the context of our research, we decided to apply a basic evolutionary algorithm instead of more advanced meta-heuristic approaches. The reason for this choice is that our goal is to lay the ground for mining process model variants instead of improving their performance.

10.3 Web Services

Web services can be considered as "self-contained, self-describing, modular applications that can be published, located, and invoked across the Web" [139]. In this context, an increasing number of companies and organizations focus on implementing their core business and outsource other application services over the Internet; i.e., they utilize web services as offered by global providers. This trend has attracted both academic and industrial researchers to look at different aspects of services including their composition, orchestration and monitoring. In this section, we focus on techniques for service composition and service monitoring.

10.3.1 Service Composition

Service composition is one of the most popular research topics in the field of web services and service-oriented computing [130, 139, 28, 118]. Service composition

refers to the combined use of existing (web) services in order to serve the business needs of a company best [139]. It often employs process management technology (e.g., process engines based on WS-BPEL (Business Process Execution Language) [21]) to orchestrate a collection of services in a process-oriented way in order to realize a particular business goal.

A a service composition is often transformed into a formal representation (e.g., a Petri Net [74, 126] or state automaton [230]) in order to apply different kinds of analyses. For example, based on such transformation we can validate, diagnose and evaluate the performance of a service composition schema (e.g., response time and failure rate) by using various kind of process analyses [194]. Respective techniques focus on the structural and behavioral aspects of a service composition schema. Furthermore several other aspects have been discussed in order to guarantee well functioning service composition schemes. For example, [175] discusses semantic aspects of service compositions and suggests a richer description of services and a more effective way for allocating resources. The approach described in [26], in turn, deals with quality aspects of service compositions; i.e., it addresses the question how a service composition schema can satisfy certain quality properties like low response time and costs.

Service choreographies further extend the scope of service orchestrations by additionally considering the interactions between multiple service compositions [225]. In this context, the Choreography Description Language (CDL) [232] was introduced for specifying service choreographies. Like composition schemes, a service choreography specification is often transformed into a formal representation in order to enable its formal verification as well as to apply performance analyses [51].

Recently, advanced techniques for dynamic service composition were introduced which allow to dynamically adapt service compositions to changes in their environment and to the needs of different customers in a simple and effective way [28, 146, 53]. Respective adaptations are similar to the ones known from adaptive process management, which has been intensively studied during the last years [133, 68, 211, 213].

When allowing for the dynamic definition and adaptation of service composition schemes, we potentially obtain a large collection of service composition schema variants derived from the same original schema, but slightly differing in their structure. Consequently, mining the schemes of these service variants becomes an interesting topic [28, 139]. For this purpose, the different techniques described in this thesis can be directly applied. Furthermore, flexibility issues with respect to service choreographies have become a subject of increasing interest during the last years. For example, the approaches described in [229, 159] allow to align choreography schema changes with BPEL schema changes.

Since many approaches on service composition and service analysis are based on process management techniques, we have provided a detailed discussion on them in Section 10.1.

10.3.2 Service Monitoring

Service monitoring techniques are used to monitor the actual behavior of service compositions and agreed upon service level agreements (SLAs) respectively [83]. Violations of these SLAs (e.g., delayed response time, lower quality of services) can be identified and punished [16, 17]. Similar techniques have been applied in business IT alignment [186], where undefined business rules or security protocols [212] are automatically identified and measured. However, most approaches for service monitoring analyze behavior inconsistencies (i.e., mismatch between the designed composition schema and its real executions) [16, 17], or data / business rule violations (i.e., deviations from pre-defined business rules or required service qualities) [6, 5]. Clearly, they have different goals as the ones of this thesis in which we focus on learning from structural control flow adaptations. As service monitoring topics are similar to conformance checking [168] or compliance checking [5] known from the field of process analysis, we have discussed further details in Section 10.1.

10.4 Algorithm Evaluation Approaches

In this thesis, we have evaluated our algorithms based on simulations (cf. Chapter 6 and Chapter 7) and case studies (cf. Chapter 9). Section 10.4.1 discusses the application of simulations in other research context. We further discuss general algorithm evaluation criteria in Section 10.4.2.

10.4.1 Simulations

Simulation is often applied in system design, analysis and evaluation, and is one of the most widely used techniques in operations research, management science and computer visualizations [95]. In a simulation, "we numerically excise the model for the inputs and see how they affect the outputs" [95]. Simulations can be used to explore and gain new insights into new algorithms or technologies, and to estimate the performance of systems which are too complex for analytical solutions [95].

In the context of this thesis, we use simulation to evaluate the performance and basic properties of our algorithms and to predict their behavior when being applied in practice and in a large scale. Such approach (i.e., using simulation to test properties of the algorithms) is not only used frequently in computer science [7, 41, 106], but also commonly applied in other fields as well. For example, Traffic engineers use simulations for the planning, design and operation of transportation systems (e.g., to plan or redesign parts of the street network from single junctions over cities to a national highway network [136]). In ecology, biologists use simulations to predict future fish populations in order to design proper fishing policies [167]. Simulation techniques are also applied by economists to evaluate the influences of certain economic regulations or policies [170].

10.4.2 Algorithm Evaluation Criteria

Besides simulation, various techniques for evaluating the performance of heuristics algorithms and clustering techniques exist. Complexity theory is often applied to evaluate time and/or space complexity of an algorithm [34]. Complexity of an algorithm is expressed in terms of polynomials to indicate how required time/space grows as the number of analyzed data-points grows [34]. However, most algorithms in our context have non-deterministic polynomial (i.e., $\mathcal{NP}$) complexity. Therefore, additional analyses of the performance of our algorithms are required.

The distance-fitness correlation, as used in this thesis, is commonly applied in the evaluation of other heuristic or genetic algorithms as well [81]. For example, [81] discusses more than 20 heuristic or genetic algorithms and uses the correlation between their fitness value and the closest local optimum to evaluate the difficulties of the problem filed and the performance of particular algorithms. [178] further uses correlation between fitness values of parent and child generations to evaluate the crossover function of a genetic algorithm.

It is also possible to evaluate an algorithm using qualitative measures. For example, we can evaluate whether or not initial centroid points can influence final clustering results, or whether a clustering algorithm is tolerant in respect to noise and outliers [181]. In addition, empirical findings or manual judgment can also play a role in algorithm evaluations. For example, in the filed of information retrieval, ranking results of a particular searching algorithm are usually compared to manually judged results. In this way they can evaluate the performance of a particular search algorithm [14, 8].

10.5 Summary

In this chapter, we have discussed existing work in the context of managing, analyzing, and mining process model variants. Particularly, we compared our algorithms with a variety of approaches in fields like traditional data analyses and data mining, process change, process similarity measurement, and process mining.

Traditional data analysis and data mining algorithms commonly do not consider important properties of a process meta model. Most process change and process analysis techniques, in turn, can take important properties of a process model into account, but do not support learning from past process changes. Though process mining algorithms support learning from past process executions and process changes, they have different inputs and goals than the algorithms presented in this thesis.

In summary, only few of the discussed approaches are appropriate for supporting the evolution of a reference process model towards an easy and cost-effective model by learning from process variants in a controlled way.

11 Summary

This thesis presented challenges, scenarios and algorithms for the representation, comparison and mining of process variants.

We first described the major goal of process variant mining. By mining a collection of process model variants, we want to discover a reference process model out of which the process variants can be easily configured, i.e., a reference process model with minimal average distance to the variants. We believe that realizing this goal will contribute to strengthen process mining tools, and enable learning from past process adaptations and process configurations, respectively. Based on an analysis of existing process mining techniques, we motivated the need for designing novel algorithms in order to achieve this goal, and we described the challenges to be tackled in this context.

We provided a matrix representation for process models, which we denote as order matrix. An order matrix captures (transitive) order relations between pairs of activities, and can be considered as unique representation of a block-structured process model. We compared the order matrix with process tree representations. The obtained results indicate that the order matrix supports process changes better and fosters the identification of process blocks. We further introduced the notion of aggregated order matrix, which aggregates a number of order matrices, in order to represent a collection of process models.

Based on order matrices we addressed the challenge to measure the efforts for process configuration. For this purpose we introduced the concept of process distance, which corresponds to the minimal number of high-level change operations needed for transforming one process model into another. In general, we assume that the shorter the average distance between a reference process model and related process variants is, the less changes are required for adapting the variants and the less efforts are needed for process configuration. In this context, we introduced a method based on boolean algebra optimization to compute the distance between two process models.

Following this, we developed, evaluated and compared two algorithms for discovering a reference process model from a collection of process variants. Adopting the discovered model as new reference process model makes (future) process configuration easier, since less efforts for configuring the variants are required.

Our clustering algorithm does not presume any knowledge of the original reference process model out of which the process variants were configured. By only looking at the process variant models, it can quickly discover a reference process model in polynomial time, which allows us to scale up when solving real-world problems. Our clustering algorithm can further provide information on how well each part of the discovered reference model fits to the variants. Such information can help to identify those regions where process configurations often occur.

Our heuristic algorithm, in turn, can also take the original reference model into account such that the user can control to what degree the discovered model is different from the original one. This way, we can avoid spaghetti-like process models and additionally control how many changes we want to perform on the original reference model. Our algorithm can automatically determine which activities shall be considered in the reference model. Filtering or pre-analysis of the activity sets are not needed in this context. We evaluated our heuristic algorithm by performing a comprehensive simulation. Based on the obtained simulation results, we can draw the following conclusions:

1. The fitness function of our heuristic search algorithm is correlated with the average weighted distance with high correlation value. This indicates good performance of our algorithm since the approximation value we use to guide the search is nicely correlated to the real one.

2. Performance of our heuristic algorithm can scale up. Its performance, which is measured based on the correlation between fitness and distance, is independent from the size of the models.

3. When discovering a new reference model by changing the original one, the more important changes, which largely reduce average weighted distance to the variants, are performed at the beginning. Our simulation results indicate that the first 1/3 of the applied changes result in about 2/3 of overall distance reduction.

Both the clustering and the heuristic algorithm have their advantages and disadvantages. The clustering algorithm has polynomial complexity; i.e., it runs significantly faster than the heuristic algorithm. In addition, our clustering algorithm reflects how each part of the discovered reference process model fits to the variants (using *separation* and *cohesion*). Our heuristic algorithm, in turn, can only provide an overall evaluation (based on *fitness*). However, the clustering algorithm cannot control the discovery procedure or distinguish important changes from less relevant ones as our heuristic algorithm does. As our simulations revealed, process models discovered by the clustering algorithm were also less optimal when compared to the ones discovered by the heuristic algorithm, since the search space of the clustering algorithm is considerably smaller.

We further compared our algorithms with traditional process mining algorithms. Based on a quantitative analysis, we showed that the reference model

discovered by our algorithm requires a lower number of change operations for configuring the variants in comparison to the models we can discover when using traditional process mining algorithms. Though behavior aspects are not considered in our algorithms, our comparison results imply that behavior of the discovered model was not sacrificed that much. Though these results cannot be generalized, they give a good motivation for integrating our algorithms with process mining techniques.

We successfully applied the two algorithms to cases from the automotive and the healthcare domain. During these studies, the practical relevance and benefit of our work became evident. Regarding the automotive case, our clustering algorithm discovered a reference process model which constitutes a good choice for representing the top level change management process. Regarding the healthcare case, our heuristic algorithm contributed to improve a clinical reference process model by significantly reducing its average weighted distance to the variants. As expected, it discovered important changes at the beginning of the search.

Though results look promising and are practically relevant, as always there is room for future research work:

- First we have to include more control structures (like synchronization constraints for parallel activities as proposed in the ADEPT framework or in WS-BPEL). Furthermore, we need to relax the constraint that labels are unique. We can also cover more practical cases if we further relax the block-structure constraint for process models. We believe that such extension can be at least partly based on the concepts and techniques presented in this thesis.
- Second, though our comparison results in Chapter 8 indicate good performance of our algorithms in both structure and behavior aspect, it would be useful to tighter integrate our algorithms with existing process mining algorithms such that we can take structure as well as behavior perspective simultaneously into account in order to cover more general cases [197].
- As learned from our case studies, data-flow and actor assignments also constitute important parts of any process configuration. Therefore, it would be advantageous to additionally consider these perspectives in our mining approach so that it cannot only consider structural control flow changes, but also include structural data flow changes [68] or changes of an organization model [112] in process configurations as well.
- Finally, we believe that process variants mining also constitutes a challenging research topic in the field of data-driven processes (i.e., processes where data and object changes drive the execution of the processes [90]). In this context there exists work targeting at the product-based support of large process structures and their coordination based on object models and relating object relations (see [123]). It would be interesting to analyze variants in that context as well.

With the increasing adoption of Process-aware Information Systems, the operational support for business processes has become an integral part of enterprise computing [225]. In particular, businesses are more and more driven by process models, which will result in the emergence of large process model repositories [164].

The ability to manage model variants in such repositories will therefore become an important issue for managing, maintaining and evolving process-aware information systems [213]. The techniques provided in this thesis constitute a good solution for managing and analyzing process variants, and therefore close a gap in respect to full process lifecycle support in PAISs. In principle, if process model variants can be transformed into a block-structured representation[135, 204, 84, 140], , our approaches become applicable for managing and analyzing large variant collections.

In the context of this thesis, we assume process models to be block-structured. On the one hand, block-structured models are easy to understand [151, 115, 116], have less chances of containing errors [114, 116, 32], facilitate advanced analysis [205, 59, 46], and are easy to change [211, 141]. These reasons explain an increasing interest of researches for also transforming non-block-structured models into block-structured ones [135, 204, 84, 140]. On the other hand, block-structured models only support a fraction of the known workflow patterns [193], i.e., expressiveness remains an issue. For the future, we are expecting a balance to be made between performance and expressiveness issues. In particular, future languages and tools should support additional patterns, while ensuring good performance as well as usability in the context of more advanced use cases like dynamic process changes.

A Appendix

A.1 Properties of Block-structured Process Model

Let $S = (A, E, AT, ET, l)$ be a block-structured process model (cf. Def. 1). Then: S has the following structural properties:

1. S has a *unique start node*; i.e.; $\exists! s \in A : \forall (a_1, a_2) \in E : a_2 \neq s$.
   s is the only node with $AT(s) = \texttt{StartFlow}$.
2. S has a *unique end node*; i.e.; $\exists! e \in A : \forall (a_1, a_2) \in E : a_1 \neq e$.
   e is the only node with $AT(e) = \texttt{EndFlow}$.
3. Let A^{types} be defined as $A^{types} := \{a \in A | AT(a) \in types\}$, Then:
   - Each *non-split node* (excl. the end node) has exactly one outgoing precedence edge:
     $\forall a \in A \setminus A^{AndSplit \bigcup XorSplit \bigcup EndFlow}$:
     $\exists! e = (a_1, a_2) \in E$ with $a_1 = a \wedge ET(e) = \texttt{Precedence}$.
   - Each *non-join node* (excl. the start node) has exactly one incoming precedence edge:
     $\forall a \in A \setminus A^{AndJoin \bigcup XorJoin \bigcup StartFlow}$:
     $\exists! e = (a_1, a_2) \in E$ with $a_2 = a \wedge ET(e) = \texttt{Precedence}$.
   - Any *loop edge* links a StartLoop node with an EndLoop node:
     $\forall e = (a_1, a_2) \in E$ with $ET(e) = \texttt{Loop}$,
     $\Rightarrow AT(a_1) = \texttt{StartLoop} \wedge AT(a_2) = \texttt{EndLoop}$.
4. S is block-structured –i.e., the following properties hold:
   - Let $Splits$, $Joins \subset A$ be defined as follows:
     $Splits := A^{AndSplit \bigcup XorSplit}$, $Joins := A^{AndJoin \bigcup XorJoin}$.
     Then: There exists a mapping $join : Splits \rightarrow Joins$ with:
     - $s \in Splits$, $\Rightarrow s \prec join(s)$.
     - $join$ is a bijective mapping, i.e., $join(s_1) = join(s_2)$ for $s_1, s_2 \in Splits$, $\Rightarrow$
       $s_1 = s_2 \wedge \forall j \in Joins : \exists s \in Splits : join(s) = j$.
     - Let $s \in Splits$:
       The subgraph induced by $\{s, join(s)\} \bigcup \{a \in A | s \prec a \wedge a \prec$

$join(s)\}$ is a SESE, i.e., a subgraph with single entry and single exit node.
  - $s \in Splits \wedge AT(s) =$ `AndSplit(XorSplit)`, $\Rightarrow AT(join(s)) =$ `AndJoin(XorJoin)`.
- There exists a bijective mapping $loop : A^{StartLoop} \rightarrow A^{EndLoop}$ with:
  - $ls \in A^{StartLoop}$, $\Rightarrow loop(ls) \prec ls$
  - $ls \in A^{StartLoop}$, $\Rightarrow$ The subgraph induced by $\{ls, loop(ls)\} \bigcup \{a \in A | loop(ls) \prec a \wedge a \prec ls\}$ is a SESE.
- Blocks must not overlap, i.e., their nesting must be regular. Formally: $B_{starts} \equiv Splits \bigcup A^{EndLoop}$; $B_{ends} \equiv Joins \bigcup A^{StartLoop}$. Further, Let *block* be a mapping,$block : B_{starts} \rightarrow B_{ends}$ with $block(s) = join(s)$ if $s \in Splits$ and $block(s) = loop^{-1}(s)$ if $s \in A^{EndLoop}$. Then: $s_1, s_2 \in B_{starts}$ with $s_1 \prec s_2 \prec block(s_1)$, $\Rightarrow block(s_2) \prec block(s_1)$.

In addition, we can define **block** as follows:

Let $a_1, a_2 \in A$ with $a_1 \prec a_2$ or $a_1 = a_2$. Let further $join(a)$ be a bijective function to map each split/startLoop node $a \in A$ with $NT(a) \in \{$`AndSplit,XorSplit,StartLoop`$\}$ to its corresponding joint/endLoop node $a' \in A$ with $NT(a') \in \{$`AndJoin,XorJoin,EndLoop`$\}$. Then: The subgraph of S induced by node set $B = \{a_1, a_2\} \bigcup \{a \in A | a_1 \prec a \wedge a \prec a_2\}$ constitutes a **block** iff:

- $\forall a \in B$ with $NT(a) \in \{$`AndSplit,XorSplit,StartLoop`$\}$, $\Rightarrow join(a) \in B$
- $\forall a \in B$ with $NT(a) \in \{$`AndJoin,XorJoin,EndLoop`$\}$, $\Rightarrow join^{-1}(a) \in B$

Note that these definitions correspond to what we have informally described in Section 2.1.1.

A.2 Proof of Theorem 1

In this appendix, we prove Theorem 1 (cf. Section 4.2). It states that we can obtain a unique order matrix A based on a given process structure tree $T = (N, C, CT, E, l)$, i.e., for any two nodes $a_i, a_j \in N$, $NCA(a_i, a_j)$ exists and is unique. The proof consists of three steps:

1. Based on the properties of process structure tree (cf. Theorem 2), we first prove that the indegree of any element in a process structure tree is less or equal to 1 (cf. Theorem 3).
2. In the second step, we show that for any two connected nodes in a process structure tree, there exists exactly one path linking them (cf. Lemma 1).
3. Finally, we prove that for any two different nodes in a process structure tree, their nearest common ancestor exists and is unique (cf. Theorem 4).

We first discuss an important property of any process structure tree $T = (N, C, CT, E, l)$, namely that a subtree of T does not overlap with another different subtree of T. This property is described by Theorem 2. For a proof of Theorem 2, we refer to [204].

Theorem 2 *Let $T = (N, C, CT, E, l)$ be a process structure tree and let $T' = (N', C', CT', E', l')$ and $T'' = (N'', C'', CT', E'', l'')$ be two different subtrees of T. Then: T' does not overlap with T''; i.e., either T' is a subtree of T'', or T'' is a subtree of T', or the following property holds $((N' \bigcap N'' = \emptyset) \wedge (C' \bigcap C'' = \emptyset) \wedge (E' \bigcap E'' = \emptyset))$.*

Based on Theorem 2, we can obtain Theorem 3.

Theorem 3 *The indegree $in(e)$ of any element $e \in N \bigcup C$ in a process structure tree $T = (N, C, CT, E, l)$ is less or equal 1.*

Proof 2 *Assume there exists an element $e \in N \bigcup C$ which has more than one predecessor; i.e., $\exists\, n_1, n_2, \ldots, n_i \in N \bigcup C : (n_1, e), (n_2, e), \ldots, (n_i, e) \in E$ and $n_x \neq n_y$ for $x, y \in \{1, \ldots, i\}$. Let $T(n_x)$ and $T(n_y)$ be two subtrees that result when using n_x and n_y as their root elements (with $x, y \in \{1, \ldots, i\}$, $x \neq y$) (cf. Section 2.1.2). Then both $T(n_x)$ and $T(n_y)$ contain element e since $(n_x, e) \in E$ and $(n_y, e) \in E$. However, since $n_x \neq n_y$ holds, $T(n_x)$ cannot be a subtree of $T(n_y)$, and vice versa. Therefore, $T(n_x)$ and $T(n_y)$ overlap, which contradicts to the property described in Theorem 2. Consequently, any element $e \in N \bigcup C$ maximally has one predecessor, i.e., $in(e) \leq 1$; $in(e) = 0$ holds if e is the root of T.*

Based on Theorem 3, we can obtain Lemma 1.

Lemma 1 *For two connected nodes $a, b \in N \bigcup C$ in a process structure tree $T = (N, C, CT, E, l)$, there exists exactly one path connecting a with b.*

Proof 3 *Let $a, b \in N \bigcup C$ be two elements of T. Assume that a and b are connected with $a \prec b$. Then there exists a sequence $< n_0, n_1, \ldots, n_i >$ with $n_0, \ldots, n_i \in N \bigcup C$, $n_0 = a$, $n_i = b$ and $(n_{k-1}, n_k) \in E$ for $k \in \{1, \ldots, i\}$. According to Theorem 3, $in(n_k) \leq 1$ holds $\Rightarrow$ The node which directly precedes n_k is unique and corresponds to n_{k-1}. Since this applies to all $k \in \{1, \ldots, i\}$, the path $< n_0, \ldots, n_i >$ is unique.*

Finally, Theorem 4 describes the existence and uniqueness of the nearest common ancestor for any two different nodes in a process structure tree.

Theorem 4 *Taking two different nodes $a, b \in N$ in a process structure tree $T = (N, C, CT, E, l)$, their nearest common ancestor $NCA(a, b)$ exists and is unique.*

Proof 4 *Let $a, b \in N$ be two different nodes and let further $c \in C$ be the nearest common ancestor of a and b. Since a and b are two different nodes, T contains at minimum two nodes and one connector. Consequently, there must be a root connector $r \in E$ with $r \prec a$ and $r \prec b$, since nodes constitute the leaves in the tree and cannot be a predecessor of any other tree element.*

- *Existence of c. Since each process structure tree has a unique root r, we obtain* $r \prec a$ *and* $r \prec b$*. According to Lemma 1, we can find two unique paths* $< n_0, n_1, \ldots, n_i >$ *with* $n_0 = r$ *and* $n_i = a$*, and* $< n'_0, n'_1, \ldots, n'_j >$ *with* $n'_0 = r$ *and* $n'_j = b$ *respectively. Let* $\min(i, j)$ *denote the minimum of* i *and* j*. Since* $r = n_0 = n'_0$*, there must be a* k *with* $0 \leq k \leq \min(i, j)$ *such that* $n_0 = n'_0$*,* $n_1 = n'_1$*, ...,* $n_k = n'_k$*. According to Def. 7 we obtain* n_k *as* $NCA(a, b)$*.*
- *Uniqueness of c. Assume that there is another connector* $c' \neq c$ *with* $c' \in C$*, which is a nearest common ancestor of a and b. According to Def. 7 we obtain* $c \not\prec c'$ *and* $c' \not\prec c$*. Let r be the unique root of process structure tree T. Then we obtain* $r \prec c$ *and* $r \prec c'$*. Since* c' *is common ancestor of a and b, we obtain* $c' \prec a$ *and* $c' \prec b$*. Consequently, there are two different paths from r to a:* $< r, \ldots, c, \ldots, a >$ *and* $< r, \ldots, c', \ldots, a >$*. This contradicts to Lemma 1* $\Rightarrow$ c' *cannot exist.*

Bibliography

[1] M. Adams, A.H.M. ter Hofstede, D. Edmond, and W.M.P. van der Aalst. Worklets: A service-oriented implementation of dynamic flexibility in workflows. In *CoopIS'06*, pages 291–308. LNCS 4275, Springer, 2006.

[2] M. Adams, A.H.M. ter Hofstede, W.M.P. van der Aalst, and D. Edmond. Dynamic, extensible and context-aware exception handling for workflows. In *CoopIS'07*, pages 95–112. LNCS 4803, Springer, 2007.

[3] A.V. Aho, M.S. Lam, R. Sethi, and J.D. Ullman. *Compilers: Principles, Techniques, and Tools (2nd Edition)*. Addison Wesley, 2006.

[4] J.S. Ash, M. Berg, and E. Coiera. Some unintended consequences of information technology in health care: the nature of patient care information system-related errors. *Journal of the American Medical Informatics Association*, 11(2):104–112, 2004.

[5] A. Awad, G. Decker, and M. Weske. Efficient compliance checking using BPMN-Q and temporal logic. In *BPM'08*, pages 326–341. LNCS 5240, Springer, 2008.

[6] A. Awad, M. Weidlich, and M. Weske. Specification, verification and explanation of violation for data aware compliance rules. In *ICSOC'09*, pages 500–515. LNCS 5900, Springer, 2009.

[7] J. Bae, L. Liu, J. Caverlee, L.J. Zhang, and H. Bae. Development of distance measures for process mining, discovery and integration. *International Journal on Web Service Research*, 4(4):1–17, 2007.

[8] R. Baeza-Yates and B Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.

[9] R.A. Baeza-Yates. Text retrieval: Theory and practice. In *Proceedings of the IFIP 12th World Computer Congress on Algorithms, Software, Architecture - Information Processing '92, Volume 1*, pages 465–476, Amsterdam, The Netherlands, The Netherlands, 1992. North-Holland Publishing Co.

[10] C. Baier and J.P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[11] P. Balabko, A. Wegmann, A. Ruppen, and N. Clément. Capturing design rationale with functional decomposition of roles in business processes modeling. *Software Process: Improvement and Practice*, 10(4):379–392, 2005.

[12] P. Bille. A survey on tree edit distance and related problems. *Theoretical Computer Science*, 337(1-3):217–239, 2005.

[13] M. Blaha and J. Rumbaugh. *Object-oriented modeling and design with UML-Second Edition*. Prentice Hall, 1991.

[14] H.M. Blanken, A.P. de Vries, H.E. Blok, and L. Feng. *Multimedia Retrieval*. Springer, 2007.

[15] R. Bobrik, M. Reichert, and T. Bauer. View-based process visualization. In *BPM'07*, pages 88–95. LNCS 4714, Springer, 2007.

[16] L. Bodenstaff, A. Wombacher, M. Reichert, and M.C. Jaeger. Monitoring dependencies for SLAs: The MoDe4SLA approach. In *SCC'08*, pages 21–29, 2008.

[17] L. Bodenstaff, A. Wombacher, M. Reichert, and M.C. Jaeger. Analyzing impact factors on composite services. In *SCC'09*, pages 218–226, 2009.

[18] S.P. Borgatti. Identifying sets of key players in a social network. *Computational & Mathematical Organization Theory*, 12(1):21–34, 2006.

[19] R.P. Jagadeesh Chandra Bose and W.M.P. van der Aalst. Context aware trace clustering: Towards improving process mining results. In *SDM'09*, pages 401–412. SIAM, 2009.

[20] A.W. Bowman, M.C. Jones, and I. Gijbels. Test monotonicity of regression. *Journal of Computational and Graphical Statistics*, 7(4):489–500, 1998.

[21] BPEL. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf.

[22] OMG BPMI.org. *Business Process Modeling Notation 2.0 Beta 2.* Object Management Group, 2010. available at: http://www.bpmn.org.

[23] S. Brown and Z. Vranesic. *Fundamentals of Digital Logic with Verilog Design.* McGraw-Hill, 2003.

[24] S. Buchwald, T. Bauer, and M. Reichert. Durchgängige Modellierung von Geschäftsprozessen in einer Service-orientierten Architektur. In *Modellierung'10*, pages 203–211. LNI 161, GI, 2010.

[25] D.S. Callaway, M.E.J. Newman, S.H. Strogatz, and D.J. Watts. Network robustness and fragility: Percolation on random graphs. *Physical Review Letter*, 85(25):5468–5471, 2000.

[26] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villan. An approach for QoS-aware service composition based on genetic algorithms. In *GECCO '05*, pages 1069–1075. ACM, 2005.

[27] F. Casati, S. Ceri, B. Pernici, and G. Pozzi. Workflow evolution. *Data & Knowledge Engineering*, 24(3):211–238, 1998.

[28] F. Casati, S. Ilnicki, LJ Jin, V. Krishnamoorthy, and M.C. Shan. Adaptive and dynamic service composition in eFlow. In *CAiSE'00*, pages 13–31. LNCS 1789, Springer, 2000.

[29] D. Chakrabarti and C. Faloutsos. Graph mining: Laws, generators, and algorithms. *ACM Computing Surveys*, 38(1):2, 2006.

[30] K.M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions on Computer Systems*, 3(1):63–75, 1985.

[31] K.M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Transactions Computer Systems*, 3(1):63–75, 1985.

[32] C. Combi and M. Gambini. Flaws in the flow: The weakness of unstructured business process modeling languages dealing with data. In *OTM Conferences (1)*, pages 42–59. LNCS 5870, Springer, 2009.

[33] J.E. Cook and A.L. Wolf. Automating process discovery through event-data analysis. In *ICSE '95*, pages 73–82. ACM, 1995.

[34] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press and McGraw-Hill Book Company, 2001.

[35] J. Cortadella, M. Kishinevsky, L. Lavagno, and A. Yakovlev. Deriving Petri Nets for finite transition systems. *IEEE Trans. Computers*, 47(8):859–882, 1998.

[36] O. Coudert. Doing two-level logic minimization 100 times faster. In *SODA '95*, pages 112–121. Society for Industrial and Applied Mathematics, 1995.

[37] P. Dadam and M. Reichert. The ADEPT project: A decade of research and development for robust and flexible process support - challenges and achievements. *Computer Science - Research and Development*, 23(2):81–97, 2009.

[38] T.H. Davenport. *Mission Critical - Realizing the Promise of Enterprise Systems*. Harvard Business School, 2000.

[39] A.K. Alves de Medeiros. *Genetic Process Mining*. PhD thesis, Eindhoven University of Technology, NL, 2006.

[40] J. Dehnert and R. Rittgen. Relaxed soundness of business processes. In *CAiSE '01*, pages 157–170, London, UK, 2001. LNCS 2068, Springer.

[41] A. Demers, S. Keshav, and S. Shenker. Analysis and simulation of a fair queueing algorithm. In *SIGCOMM '89*, pages 1–12. ACM, 1989.

[42] I.S. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning. In *KDD '01*, pages 269–274, New York, NY, USA, 2001. ACM.

[43] R.M. Dijkman, M. Dumas, and L. García-Bañuelos. Graph matching algorithms for business process model similarity search. In *BPM'09*, pages 48–63. LNCS 5701, Springer, 2009.

[44] R.M. Dijkman, M. Dumas, L. Garcia-Banuelos, and R. Kaarik. Aligning business process models. In *EDOC'09*, pages 45–53, 2009.

[45] E.W. Dijkstra. Notes on structured programming. pages 1–82, 1972.

[46] R. Eshuis and P.W.P.J. Grefen. Structural matching of BPEL processes. In *ECOWS'07*, pages 171–180. IEEE Computer Society, 2007.

[47] B. D. Estrade, L. A. Perkins, and J. M. Harris. Explicitly parallel regular expressions. In *IMSCCS'06*, pages 402–409. IEEE Computer Society, 2006.

[48] D.R. Ferreira and D. Gillblad. Discovering process models from unlabelled event logs. In *BPM'09*, pages 143–158. LNCS 5701, Springer, 2009.

[49] D.R. Ferreira, M. Zacarias, M. Malheiros, and P. Ferreira. Approaching process mining with sequence clustering: Experiments and findings. In *BPM'07*, pages 360–374. LNCS 4714, Springer, 2007.

[50] H.M. Ferreira and D.R. Ferreira. An integrated life cycle for workflow management based on learning and planning. *International Journal on Cooperative Information Systems*, 15(4):485–505, 2006.

[51] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Compatibility verification for web service choreography. In *ICWS'04*, pages 738–741. IEEE Computer Society, 2004.

[52] A. Ghose and G. Koliadis. Auditing business process compliance. In *ICSOC'07*, pages 169–180. LNCS 4749, Springer, 2007.

[53] J. Gordijn, H. Weigand, M. Reichert, and R. Wieringa. Towards self-configuration and management of e-service provisioning in dynamic value constellations. In *SAC'08*, pages 566–571. ACM, 2008.

[54] F. Gottschalk. *Configurable Process Models*. PhD thesis, Eindhoven University of Technology, The Netherlands, December 2009.

[55] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable workflow models. *International Journal on Cooperative Information Systems*, 17(2):177–221, 2008.

[56] F. Gottschalk, T.A.C. Wagemakers, M.H. Jansen-Vullers, W.M.P. van der Aalst, and M. La Rosa. Configurable process models: Experiences from a municipality case study. In *CAiSE'09*, pages 486–500. LNCS 5565, Springer, 2009.

[57] G. Greco, A. Guzzo, and L. Pontieri. Mining hierarchies of models: From abstract views to concrete specifications. In *BPM'05*, pages 32–47. LNCS 3649, Springer, 2005.

[58] D. Grigori, F. Casati, U. Dayal, and M. Shan. Improving business process quality through exception understanding, prediction, and prevention. In *VLDB '01*, pages 159–168, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[59] T. Gschwind, J. Koehler, and J. Wong. Applying patterns during business process modeling. In *BPM'08*, pages 4–19. LNCS 5240, Springer, 2008.

[60] C. Günther. *Process Mining in Flexible Environments*. PhD thesis, Technical University of Eindhoven, 2009.

[61] C.W. Günther, S. Rinderle, M. Reichert, and W.M.P. van der Aalst. Change mining in adaptive process management systems. In *CoopIS'06*, pages 309–326, 2006.

[62] C.W. Günther, S. Rinderle-Ma, M. Reichert, W.M.P. van der Aalst, and J. Recker. Using process mining to learn from process changes in evolutionary systems. *Int'l Journal of Business Process Integration and Management*, 3(1):61–78, 2008.

[63] C.W. Günther, A. Rozinat, and W.M.P. van der Aalst. Activity mining by global trace segmentation. In *BPI'09*, pages 128–139. LNBIP 43, 2009.

[64] C.W. Günther and W.M.P. van der Aalst. A generic import framework for process event logs. In *BPI'06*, pages 81–92. LNCS 4103, Springer, 2006.

[65] A. Hallerbach, T. Bauer, and M. Reichert. Context-based configuration of process variants. In *TCoB'08*, pages 31–40, 2008.

[66] A. Hallerbach, T. Bauer, and M. Reichert. Managing process variants in the process lifecycle. In *ICEIS '08*, pages 154–161. Springer, 2008.

[67] A. Hallerbach, T. Bauer, and M. Reichert. Guaranteeing soundness of configurable process variants in provop. In *CEC'09*, pages 98–105, 2009.

[68] A. Hallerbach, T. Bauer, and M. Reichert. Capturing variability in business process models: the Provop approach. *Journal of Software Maintenance and Evolution: Research and Practice*, 22(6-7):519–546, 2010.

[69] G. Halmans and K. Kohl. Communicating the variability of a software product family to customers. *Software and Systems Modeling*, 2(1):15–36, 2003.

[70] D. Harel and R.E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[71] H.J. Harrington. *Business Process Improvement: The Breakthrough Strategy for Total Quality, Productivity, and Competitiveness.* McGraw-Hill, 1991.

[72] H. Hermanns, J.P. Katoen, J. Meyer-Kayser, and M. Siegle. A tool for model-checking markov chains. *International Journal on Software Tools for Technology Transfer*, 4(2):153–172, 2003.

[73] J. Hidders, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, and J. Verelst. When are two workflows the same? In *CATS '05:*, pages 3–11, Darlinghurst, Australia, 2005.

[74] S. Hinz, K. Schmidt, and C. Stahl. Transforming BPEL to Petri Nets. In *BPM'05*, pages 220–235. LNCS 3649, Springer, 2005.

[75] L.B. Holder, D.J. Cook, and S. Djoko. Substructure discovery in the subdue system. In *In Proc. of the AAAI Workshop on Knowledge Discovery in Databases*, pages 169–180, 1994.

[76] C.P. Holland and B. Light. A critical success factors model for ERP implementation. *IEEE Software*, 16(3):30–36, 1999.

[77] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation (3rd Edition).* Addison Wesley, 2007.

[78] J. Huan, W. Wang, J. Prins, and J. Yang. SPIN: mining maximal frequent subgraphs from graph databases. In *KDD '04*, pages 581–586, New York, NY, USA, 2004. ACM.

[79] M.H. Jansen-Vullers, W.M.P. van der Aalst, and M. Rosemann. Mining configurable enterprise information systems. *Data Knowl. Eng.*, 56(3):195–244, 2006.

[80] T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. In *CPM '94*, pages 75–86. Springer-Verlag, 1994.

[81] T. Jones and S. Forrest. Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In *6th International Conference on Genetic Algorithms*, pages 184–192, USA, 1995. Morgan Kaufmann.

[82] G. Karypis, EH. Han, and V. Kumar. Chameleon: Hierarchical clustering using dynamic modeling. *Computer*, 32(8):68–75, 1999.

[83] A. Keller and H. Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *J. Network and System Management*, 11(1):57–81, 2003.

[84] B. Kiepuszewski, A.H.M. ter Hofstede, and C. Bussler. On structured workflow modelling. In *CAiSE'00*, pages 431–445. LNCS 1789, Springer, 2000.

[85] S. Kirkpatrick, C.D. Gelatt Jr., and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[86] J. Köbler, U. Schöning, and J. Torán. *The graph isomorphism problem: its structural complexity*. Birkhauser Verlag, 1993.

[87] M. Kradolfer and A. Geppert. Dynamic workflow schema evolution based on workflow type versioning and workflow migration. In *CoopIS '99*, pages 104 – 114, Washington, DC, USA, 1999. IEEE Computer Society.

[88] P. Kruchten. *The Rational Unified Process: An Introduction, Second Edition*. Addison-Wesley, 2000.

[89] R.L. Kruse and A.J. Ryba. *Data Structures And Programming Design in C++*. Prentice Hall, 1999.

[90] V. Künzle and M. Reichert. Integrating users in object-aware process management systems: Issues and challenges. In *BPMDS'09*, pages 29–41. LNBIP 43, Springer, 2009.

[91] M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *ICDM'01*, pages 313–320. IEEE Computer Society, 2001.

[92] J.M. Küster, C. Gerth, A. Förster, and G. Engels. Detecting and resolving process model differences in the absence of a change log. In *BPM'08*, pages 244–260. LNCS 5240, Springer, 2008.

[93] W. Labio and H. Garcia-Molina. Efficient snapshot differential algorithms for data warehousing. In *VLDB '96*, pages 63–74, San Francisco, CA, USA, 1996.

[94] A. Lanz, U. Kreher, M. Reichert, and P. Dadam. Enabling process support for advanced applications with the AristaFlow BPM Suite. In *BPM'10 Demonstration Track*. CEUR Workshop Proceedings, Vol. 615, 2010.

[95] A.M. Law. *Simulation modeling and analysis*. McGraw-Hill Higher Education, 2006.

[96] R. Lenz and M. Reichert. IT support for healthcare processes - premises, challenges, perspectives. *Data Knowledge Engineering*, 61(1):39–58, 2007.

[97] V. Jorge Leon and R. Balakrishnan. Strength and adaptability of problem-space based neighborhoods for resource-constrained scheduling. *OR Spectrum*, 17(1-3):173–182, 1995.

[98] C. Li, M. Reichert, and A. Wombacher. Discovering reference process models by mining process variants. In *ICWS'08*, pages 45–53. IEEE Computer Society, 2008.

[99] C. Li, M. Reichert, and A. Wombacher. Mining process variants: Goals and issues. In *IEEE SCC (2)*, pages 573–576. IEEE Computer Society, 2008.

[100] C. Li, M. Reichert, and A. Wombacher. On measuring process model similarity based on high-level change operations. In *ER '08*, pages 248–262. LNCS 5231, Springer, 2008.

[101] C. Li, M. Reichert, and A. Wombacher. Discovering reference models by mining process variants using a heuristic approach. In *BPM'09, LNCS 5701*, pages 344–362. Springer, 2009.

[102] C. Li, M. Reichert, and A. Wombacher. A heuristic approach for discovering reference models by mining process model variants. Technical Report TR-CTIT-09-08, University of Twente, The Netherlands, March 2009.

[103] C. Li, M. Reichert, and A. Wombacher. Representing block-structured process models as order matrices: Basic concepts, formal properties, algorithms. Technical Report TR-CTIT-09-47, University of Twente, The Netherlands, 2009.

[104] C. Li, M. Reichert, and A. Wombacher. What are the problem makers: Ranking activities according to their relevance for process changes. In *ICWS'09*, pages 51–58. IEEE Computer Society, 2009.

[105] C. Li, M. Reichert, and A. Wombacher. MinAdept - the clustering approach for discovering reference models out of process variants. *International Journal of Cooperative Information Systems*, 19(3), 2010.

[106] C. Li, J.M. van den Akker, S. Brinkkemper, and G. Diepen. An integrated approach for requirement selection and scheduling in software release planning. *Requirements Engineering Journal*, to appear.

[107] R. Lu, S. Sadiq, and G. Governatori. On managing business processes variants. *Data Knowledge Engineering*, 68(7):642–664, 2009.

[108] R. Lu and S. W. Sadiq. On the discovery of preferred work practice through business process variants. In *ER*, pages 165–180. Springer, 2007.

[109] R. Lu, S.W. Sadiq, V. Padmanabhan, and G. Governatori. Using a temporal constraint network for business process execution. In *ADC'06*, pages 157–166. Australian Computer Society, 2006.

[110] G. F. Luger. *Artificial Intelligence: Structures and Strategies for Complex Problem Solving*. Pearson Education, 2005.

[111] J.C.S. Lui and M.F. Chan. An efficient partitioning algorithm for distributed virtual environment systems. *IEEE Transactions on Parallel & Distributed Systems*, 13(3):193–211, 2002.

[112] L. Thao Ly, S. Rinderle, P. Dadam, and M. Reichert. Mining staff assignment rules from event-based data. In *BPI'05*, pages 177–190. LNCS 3812, Springer, 2005.

[113] J. Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction and Guidelines for Correctness*, volume 6 of *LNBIP*. Springer, 2008.

[114] J. Mendling, G. Neumann, and W.M.P. van der Aalst. Understanding the occurrence of errors in process models based on metrics. In *CoopIS'07, LNCS 4803*, pages 113–130, 2007.

[115] J. Mendling, H.A. Reijers, and J. Cardoso. What makes process models understandable? In *BPM'07*, pages 48–63. LNCS 4714, Springer, 2007.

[116] J. Mendling, H.A. Reijers, and W.M.P. van der Aalst. Seven process modeling guidelines (7pmg). *Information & Software Technology*, 52(2):127–136, 2010.

[117] J. Mendling, B.F. van Dongen, and W.M.P. van der Aalst. Getting rid of or-joins and multiple start events in business process models. *Enterprise Information Systems*, 2(4):403–419, 2008.

[118] N. Milanovic and M. Malek. Current solutions for web service composition. *IEEE Internet Computing*, 8(6):51–59, 2004.

[119] M. Minor, A. Tartakovski, and D. Schmalenand R. Bergmann. Agile workflow technology and case-based change reuse for long-term processes. *International Journal of Intelligent Information Technologies*, 4(1):80–98, 2008.

[120] A. Mishchenko, B. Steinbach, and M. Perkowski. An algorithm for bi-decomposition of logic functions. In *DAC '01*, pages 103–108. ACM, 2001.

[121] J. Moody. Race, school integration, and friendship segregation in america. *American Journal Of Sociology*, 107(3):679–716, 2001.

[122] D. Müller, J. Herbst, M. Hammori, and M. Reichert. IT support for release management processes in the automotive industry. In *BPM'06*, pages 368–377. LNCS 4102, Springer, 2006.

[123] D. Müller, M. Reichert, and J. Herbst. A new paradigm for the enactment and dynamic adaptation of data-driven process structures. In *CAiSE'08*, pages 48–63. LNCS 5074, Springer, 2008.

[124] R. Müller, U. Greiner, and E. Rahm. AGENT WORK: a workflow system supporting rule-based workflow adaptation. *Data Knowledge Engineering*, 51(2):223–256, 2004.

[125] N. A. Mulyar. *Patterns for process-aware information systems:an approach based on colored Petri nets*. PhD thesis, Technische Universiteit Eindhoven, 2009.

[126] T. Murata. Petri Nets: Properties, analysis and applications. *IEEE*, 77(4):541–580, 1989.

[127] B. Mutschler, J. Bumiller, and M. Reichert. Why process-orientation is scarce: An empirical study of process-oriented information systems in the automotive industry. In *EDOC'06*, pages 433–440. IEEE Computer Society, 2006.

[128] B. Mutschler, M. Reichert, and J. Bumiller. Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors and implications. *IEEE Transactions on Systems, Man, and Cybernetics (Part C)*, 38(3):280–291, 2008.

[129] B. Mutschler, B. Weber, and M. Reichert. Workflow management versus case handling: results from a controlled software experiment. In *SAC'08*, pages 82–89. ACM, 2008.

[130] M.P. Papazoglou and W.J. Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, 2007.

[131] D.L. Parnas. Software aging. In *ICSE '94*, pages 279–287. IEEE Computer Society Press, 1994.

[132] C. Peltz. Web services orchestration and choreography. *Computer*, 36:46–52, 2003.

[133] M. Pesic, M.H. Schonenberg, N. Sidorova, and W.M.P. van der Aalst. Constraint-based workflow models: Change made easy. In *CoopIS'07*, pages 77–94. LNCS 4803, Springer, 2007.

[134] S.S. Pinter and M. Golani. Discovering workflow models from activities' lifespans. *Comput. Ind.*, 53(3):283–296, 2004.

[135] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring acyclic process models. In *BPM'09*, pages 276–293. LNCS 6336, Springer, 2010.

[136] Y. QI and H.N. Koutsopoulos. A microscopic traffic simulator for evaluation of dynamic traffic management systems. *Transportation Research*, 4(3):113 – 129, 1996.

[137] S. Quaglini, M. Stefanelli, G. Lanzola, V. Caporusso, and S. Panzarasa. Flexible guideline-based patient careflow systems. *Artificial Intelligence in Medicine*, 22(1):65–80, 2001.

[138] J. Ross Quinlan. *C4.5: programs for machine learning*. Morgan Kaufmann Publishers Inc., USA, 1993.

[139] J. Rao and X. Su. A survey of automated web service composition methods. In *SWSWPC'04*, pages 43–54. LNCS 3387, Springer, 2004.

[140] M. Reichert. *Dynamische Ablaufänderungen in Workflow-Management-Systemen*. PhD thesis, Ulm University, Germany, 2000.

[141] M. Reichert and P. Dadam. ADEPTflex - supporting dynamic changes of workflows without losing control. *Journal of Intelligent Information Systems*, 10(2):93–129, 1998.

[142] M. Reichert, P. Dadam, and T. Bauer. Dealing with forward and backward jumps in workflow management systems. *Software and System Modeling*, 2(1):37–58, 2003.

[143] M. Reichert, P. Dadam, U. Kreher, M. Jurisch, and K. Göser. Architectural design of flexible process management technology. In *PRIMIUM Subconference at the Multikonferenz Wirtschaftsinformatik (MKWI)*. CEUR Workshop Proceedings 328, 2008.

[144] M. Reichert, P. Dadam, S. Rinderle-Ma, M. Jurisch, U. Kreher, and K. Göser. Architectural principles and components of adaptive process management technology. In *PRIMIUM'09*, pages 81–97. LNI P-151, 2009.

[145] M. Reichert, P. Dadam, S. Rinderle-Ma, A. Lanz, R. Pryss, M. Predeschly, J. Kolb, L.T. Ly, M. Jurisch, U. Kreher, and K. Goeser. Enabling Poka-Yoke workflows with the AristaFlow BPM Suite. In *BPM'09 Demonstration Track*. CEUR Workshop Proceedings, Vol. 489, 2009.

[146] M. Reichert and S. Rinderle. On design principles for realizing adaptive service flows with BPEL. In *EMISA'06*, pages 133–146. LNI 95, Koellen, 2006.

[147] M. Reichert, S. Rinderle, and P. Dadam. On the common support of workflow type and instance changes under correctness constraints. In *CoopIS'03*, pages 407–425. LNCS 2888, Springer, 2003.

[148] M. Reichert, S. Rinderle, U. Kreher, and P. Dadam. Adaptive process management with ADEPT2. In *ICDE '05*, pages 1113–1114. IEEE Computer Society, 2005.

[149] M. Reichert, S. Rinderle-Ma, and P. Dadam. Flexibility in process-aware information systems. *LNCS Transactions Petri Nets and Other Models of Concurrency*, 2:115–135, 2009.

[150] H.A. Reijers and W.M.P. Aalst. The effectiveness of workflow management systems: predictions and lessons learned. *Int'l Journal of Information Management*, 25(5):457–471, 2005.

[151] H.A. Reijers and J. Mendling. Modularity in process models: Review and effects. In *BPM'08*, pages 20–35. LNCS 5240, Springer, 2008.

[152] S. Rinderle. *Schema Evolution in Process Management Systems*. PhD thesis, Ulm University, Germany, 2004.

[153] S. Rinderle, M. Jurisch, and M. Reichert. On deriving net change information from change logs - the DELTALAYER-algorithm. In *BTW'07*, pages 364–381, 2007.

[154] S. Rinderle, M. Reichert, and P. Dadam. Correctness criteria for dynamic changes in workflow systems – a survey. *Data and Knowledge Engineering*, 50(1):9–34, 2004.

[155] S. Rinderle, M. Reichert, and P. Dadam. Flexible support of team processes by adaptive workflow systems. *Distributed and Parallel Databases*, 16(1):91–116, 2004.

[156] S. Rinderle, M. Reichert, and P. Dadam. On dealing with structural conflicts between process type and instance changes. In *BPM'04*, pages 274–289. LNCS 3080, Springer, 2004.

[157] S. Rinderle, M. Reichert, M. Jurisch, and U. Kreher. On representing, purging, and utilizing change logs in process management systems. In *BPM'06*, pages 241–256. LNCS 4102, Springer, 2006.

[158] S. Rinderle, B. Weber, M. Reichert, and W. Wild. Integrating process learning and process evolution - a semantics based approach. In *BPM'05*, LNCS 3649, pages 252–267, 2006.

[159] S. Rinderle, A. Wombacher, and M. Reichert. Evolution of process choreographies in DYCHOR. In *CoopIS'06*, pages 273–290. LNCS 4275, Springer, 2006.

[160] S. Rinderle-Ma, M. Reichert, and B. Weber. On the formal semantics of change patterns in process-aware information systems. In *ER'08*, LNCS 5231, pages 279–293, 2008.

[161] L. Rising and J.S. Norman. The scrum software development process for small teams. *IEEE Software*, 17(4):26–32, 2000.

[162] M. La Rosa. *Managing Variability in Process-Aware Information Systems*. PhD thesis, Queensland University of Technology, Australia, April 2009.

[163] M. La Rosa, W.M.P. van der Aalst, M. Dumas, and A.H.M. ter Hofstede. Questionnaire-based variability modeling for system configuration. *Software and System Modeling*, 8(2):251–274, 2009.

[164] M. Rosemann. Potential pitfalls of process modeling: Part B. *Business Process Management Journal*, 12(3):127–136, 2006.

[165] M. Rosemann and W.M.P. van der Aalst. A configurable reference modelling language. *Information Systems*, 32(1):1–23, 2007.

[166] K.H. Rosen. *Discrete Mathematics and Its Application*. McGraw-Hill, 2003.

[167] R. J. Rowley. Marine reserves in fisheries management. *Aquatic Conservation: Marine and Freshwater Ecosystems*, 4(3):233–254, 2006.

[168] A. Rozinat and W.M.P. van der Aalst. Conformance checking of processes based on monitoring real behavior. *Information Systems*, 33(1):64–95, 2008.

[169] R. Sabherwal and Y.E. Chan. Alignment between business and is strategies: A study of prospectors, analyzers, and defenders. *Information Systems Research*, 12(1):11–33, 2001.

[170] E. Sadoulet and A. de Janvry. *Quantitative Development Policy Analysis*. The Johns Hopkins University Press, 1995.

[171] A.W. Scheer. *ARIS - Business Process Modeling*. Springer, 2000.

[172] G. Schimm. Process miner - a tool for mining process schemes from event-based data. In *JELIA '02*, pages 525–528. LNCS 2424, Springer, 2002.

[173] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. Prentice Hall, 2001.

[174] D.J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, 2004.

[175] K. Sivashanmugam, J.A. Miller, A.P. Sheth, and K. Verma. Framework for semantic web process composition. *International Journal of Electronic Commerce*, 9(2):71–106, 04-5.

[176] S.F. Smith. *A learning system based on genetic adaptive algorithms*. PhD thesis, University of Pittsburgh, USA, 1980.

[177] M. Song, C. W. Günther, and W. M. P. van der Aalst. Trace clustering in process mining. In *BPI'08*, pages 109–120. LNBIP 17, Springer, 2008.

[178] P. Spiessens and B. Manderick. Finding optimal representations using the crossover correlation coefficient. In *SBIA '96*, pages 91–100. LNCS 1159, Springer, 1996.

[179] SPSS. http://www.spss.com/.

[180] KC. Tai. The tree-to-tree correction problem. *J. ACM*, 26(3):422–433, 1979.

[181] P.N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[182] L. Thom, M. Reichert, and C. Iochpe. Activity patterns in process-aware information systems: Basic concepts and empirical evidence. *International Journal of Business Process Integration and Management*, 4(2):93–110, 2009.

[183] W. M. P. van der Aalst, M. Dumas, F. Gottschalk, A. H. M. ter Hofstede, M. La Rosa, and J. Mendling. Preserving correctness during business process model configuration. *Formal Aspects of Computing*, 22(3-4):459–482, 2010.

[184] W. M. P. van der Aalst, N. Lohmann, M. La Rosa, and J. Xu. Correctness ensuring process configuration: An approach based on partner synthesis. In *BPM'10*, pages 95–111. LNCS 6336, Springer, 2010.

[185] W.M.P. van der Aalst. The application of Petri Nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66, 1998.

[186] W.M.P. van der Aalst. Business alignment: using process mining as a tool for delta analysis and conformance testing. *Requirement Engineering Journal*, 10(3):198–211, 2005.

[187] W.M.P. van der Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theoretical Computer Science*, 270(1-2):125–203, 2002.

[188] W.M.P. van der Aalst, A.K.A. de Medeiros, and A.J.M.M. Weijters. Process equivalence: Comparing two process models based on observed behavior. In *BPM'06*, pages 129–144. LNCS 4102, Springer, 2006.

[189] W.M.P. van der Aalst, M. Pesic, and H. Schonenberg. Declarative workflows: Balancing between flexibility and support. *Computer Science - Research & Development*, 23(2):99–113, 2009.

[190] W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering social networks from event logs. *Comput. Supported Coop. Work*, 14(6):549–593, 2005.

[191] W.M.P. van der Aalst and T.Basten. Identifying commonalities and differences in object life cycles using behavioral inheritance. In *ICATPN '01*, pages 32–52. Springer, 2001.

[192] W.M.P. van der Aalst and A.H.M. ter Hofstede. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, 2005.

[193] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A. P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003.

[194] W.M.P. van der Aalst, A.H.M. ter Hofstede, and M. Weske. Business process management: A survey. In *BPM'03*, pages 1–12. LNCS 2678, Springer, 2003.

[195] W.M.P. van der Aalst, B.F. van Dongen, J. Herbst, L. Maruster, G. Schimm, and A.J.M.M. Weijters. Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.*, 47(2):237–267, 2003.

[196] W.M.P. van der Aalst and K. van Hee. *Workflow Management: Models, Methods, and Systems.* The MIT Press, 2002.

[197] W.M.P van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Trans. on Knowl. and Data Eng.*, 16(9):1128–1142, 2004.

[198] W.M.P. van der Aalst, M. Weske, and D. Grünbauer. Case handling: a new paradigm for business process support. *Data Knowledge Engineering*, 53(2):129–162, 2005.

[199] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. In *Petri Nets*, pages 368–387. LNCS 5062, Springer, 2008.

[200] B. F. van Dongen and W.M.P. van der Aalst. Multi-phase process mining: Building instance graphs. In *ER'04*, pages 362–376. LNCS 3288, Springer, 2004.

[201] B.F. van Dongen, A.K.A. de Medeiros, H.M.W. Verbeek, A.J.M.M. Weijters, and W.M.P. van der Aalst. The ProM framework: A new era in process mining tool support. In *ICATPN'05*, pages 444–454. LNCS 3536, Springer, 2005.

[202] B.F. van Dongen, R. M. Dijkman, and J. Mendling. Measuring similarity between business process models. In *CAiSE'08*, pages 450–464. LNCS 5074, Springer, 2008.

[203] R.J. van Glabbeek and W.P. Weijland. Branching time and abstraction in bisimulation semantics. *Journal of ACM*, 43(3):555–600, 1996.

[204] J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. *Data Knowledge Engineering*, 68(9):793–818, 2009.

[205] J. Vanhatalo, H. Völzer, and F. Leymann. Faster and more focused control-flow analysis for business process models through sese decomposition. In *ICSOC'07*, pages 43–55. LNCS 4749, Springer, 2007.

[206] G. M. Veiga and D. R. Ferreira. Understanding spaghetti models with sequence clustering for prom. In *BPI'09*, pages 92–103. LNBIP 43, 2010, 2009.

[207] G. Vossen and M. Weske. The WASA2 object-oriented workflow management system. In *SIGMOD'99*, pages 587–589, 1999.

[208] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *J. ACM*, 21(1):168–173, 1974.

[209] B. Weber, B. Mutschler, and M. Reichert. Investigating the effort of using business process management technology: Results from a controlled experiment. *Science of Computer Programming*, 75(5):292–310, 2010.

[210] B. Weber and M. Reichert. Refactoring process models in large process repositories. In *CAiSE'08*, pages 124–139. LNCS 5074, Springer, 2008.

[211] B. Weber, M. Reichert, and S. Rinderle-Ma. Change patterns and change support features - enhancing flexibility in process-aware information systems. *Data and Knowledge Engineering*, 66(3):438–466, 2008.

[212] B. Weber, M. Reichert, W. Wild, and S. Rinderle. Balancing flexibility and security in adaptive process management systems. In *CooplS '05*, pages 59–76, 2005.

[213] B. Weber, M. Reichert, W. Wild, and S. Rinderle-Ma. Providing integrated life cycle support in process-aware information systems. *International Journal of Cooperative Information Systems*, 19(1):115–165, 2009.

[214] B. Weber, H.A. Reijers, S. Zugal, and W. Wild. The declarative approach to business process execution: An empirical test. In *CAiSE'09*, pages 470–485. LNCS 5565, Springer, 2009.

[215] B. Weber, S. Rinderle, and M. Reichert. Change patterns and change support features in process-aware information systems. In *CAiSE'07*, pages 574–588, 2007.

[216] B. Weber, S. Rinderle, W. Wild, and M. Reichert. CCBR-driven business process evolution. In *ICCBR'05*, pages 610–624. LNCS 3620, Springer, 2005.

[217] B. Weber, S. Sadiq, and M. Reichert. Beyond rigidity - dynamic process lifecycle support: A survey on dynamic changes in process-aware information systems. *Computer Science - R&D*, 23(2):47–65, 2009.

[218] B. Weber, W. Wild, M. Lauer, and M. Reichert. Improving exception handling by discovering change dependencies in adaptive process management systems. In *BPM'06 Workshops*, pages 93–104. LNCS 4103, Springer, 2006.

[219] M. Weidlich and M. Weske. Structural and behavioural commonalities of process variants. In *ZEUS'10*, pages 41–48. CEUR Workshop Proceedings 563, 2010.

[220] M. Weidlich, Ma. Weske, and J. Mendling. Change propagation in process models using behavioural profiles. In *SCC'09*, pages 33–40. IEEE, 2009.

[221] A.J.M.M. Weijters and W.M.P. van der Aalst. Rediscovering workflow models from event-based data using little thumb. *Integr. Comput.-Aided Eng.*, 10(2):151–162, 2003.

[222] L. Wen, J. Wang, and J. Sun. Detecting implicit dependencies between tasks from event logs. In *APWeb'06*, pages 591–603, 2006.

[223] M. Weske. Workflow management systems: Formal foundation, conceptual design, implementation aspects. *Habilitationsschrift Fachbereich Mathematik und Informatik, Universität Münster*, 2000.

[224] M. Weske. Formal foundation and conceptual design of dynamic adaptations in a workflow management system. In *HICSS '01*, page 7051, Washington, DC, 2001.

[225] M. Weske. *Business Process Management*. Springer, 2007.

[226] R.J. Wieringa and J.M.G. Heerkens. The methodological soundness of requirements engineering papers: a conceptual framework and two case studies. *Requirement Engineering*, 11(4):295–307, 2006.

[227] R.J. Wieringa, N.A.M. Maiden, N.R. Mead, and C. Rolland. Requirements engineering paper classification and evaluation criteria: a proposal and a discussion. *Requirement Engineering*, 11(1):102–107, 2005.

[228] I.H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann Publishers Inc., USA, 2005.

[229] A. Wombacher. Alignment of choreography changes in BPEL processes. In *SCC'09*, pages 1–8. IEEE Computer Society, 2009.

[230] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming BPEL into annotated deterministic finite state automata for service discovery. *Web Services, IEEE International Conference on*, 0:316, 2004.

[231] A. Wombacher and M. Rozie. Evaluation of workflow similarity measures in service discovery. *Service Oriented Electronic Commerce*, pages 51–71, 2006.

[232] WS-CDL. http://www.w3.org/tr/ws-cdl-10/.

[233] WSDL. http://www.w3.org/tr/wsdl.

[234] XML. http://www.w3.org/tr/rec-xml/.

[235] X. Yan and J. Han. gSpan: Graph-based substructure pattern mining. In *ICDM'02*, pages 721–724. IEEE Computer Society, 2002.

[236] M. zur Muehlen and J. Recker. How much language is enough? theoretical and practical use of the business process modeling notation. In *CAiSE'08*, pages 465–479. LNCS 5074, Springer, 2008.

[237] M. zur Muehlen and M. Rosemann. Workflow-based process monitoring and controlling - technical and organizational issues. In *HICSS '00*, page 6032, Washington, DC, USA, 2000. IEEE Computer Society.

SIKS Dissertatiereeks

====
1998
====

1998-1 Johan van den Akker (CWI)
DEGAS - An Active, Temporal Database of Autonomous Objects

1998-2 Floris Wiesman (UM)
Information Retrieval by Graphically Browsing Meta-Information

1998-3 Ans Steuten (TUD)
A Contribution to the Linguistic Analysis of Business Conversations within the Language/Action Perspective

1998-4 Dennis Breuker (UM)
Memory versus Search in Games

1998-5 E.W.Oskamp (RUL)
Computerondersteuning bij Straftoemeting

====
1999
====

1999-1 Mark Sloof (VU)
Physiology of Quality Change Modelling;
Automated modelling of Quality Change of Agricultural Products

1999-2 Rob Potharst (EUR)
Classification using decision trees and neural nets

1999-3 Don Beal (UM)
The Nature of Minimax Search

1999-4 Jacques Penders (UM)
The practical Art of Moving Physical Objects

1999-5 Aldo de Moor (KUB)
Empowering Communities: A Method for the Legitimate User-Driven Specification of Network Information Systems

1999-6 Niek J.E. Wijngaards (VU)
Re-design of compositional systems

1999-7 David Spelt (UT)
Verification support for object database design

1999-8 Jacques H.J. Lenting (UM)
Informed Gambling: Conception and Analysis of a Multi-Agent Mechanism for Discrete Reallocation.

====
2000
====

2000-1 Frank Niessink (VU)
Perspectives on Improving Software Maintenance

2000-2 Koen Holtman (TUE)
Prototyping of CMS Storage Management

2000-3 Carolien M.T. Metselaar (UVA)
Sociaal-organisatorische gevolgen van kennistechnologie;
een procesbenadering en actorperspectief.

2000-4 Geert de Haan (VU)
ETAG, A Formal Model of Competence Knowledge for User Interface Design

2000-5 Ruud van der Pol (UM)
Knowledge-based Query Formulation in Information Retrieval.

2000-6 Rogier van Eijk (UU)
Programming Languages for Agent Communication

2000-7 Niels Peek (UU)
Decision-theoretic Planning of Clinical Patient Management

2000-8 Veerle Coup (EUR)
Sensitivity Analyis of Decision-Theoretic Networks

2000-9 Florian Waas (CWI)
Principles of Probabilistic Query Optimization

2000-10 Niels Nes (CWI)
Image Database Management System Design Considerations,
Algorithms and Architecture

2000-11 Jonas Karlsson (CWI)
Scalable Distributed Data Structures for Database Management

====
2001
====

2001-1 Silja Renooij (UU)
Qualitative Approaches to Quantifying Probabilistic Networks

2001-2 Koen Hindriks (UU)
Agent Programming Languages: Programming with Mental Models

2001-3 Maarten van Someren (UvA)
Learning as problem solving

2001-4 Evgueni Smirnov (UM)
Conjunctive and Disjunctive Version Spaces with
Instance-Based Boundary Sets

2001-5 Jacco van Ossenbruggen (VU)
Processing Structured Hypermedia: A Matter of Style

2001-6 Martijn van Welie (VU)
Task-based User Interface Design

2001-7 Bastiaan Schonhage (VU)
Diva: Architectural Perspectives on Information Visualization

2001-8 Pascal van Eck (VU)
A Compositional Semantic Structure for Multi-Agent Systems Dynamics.

2001-9 Pieter Jan 't Hoen (RUL)
Towards Distributed Development of Large Object-Oriented Models,
Views of Packages as Classes

2001-10 Maarten Sierhuis (UvA)
Modeling and Simulating Work Practice

BRAHMS: a multiagent modeling and simulation language
for work practice analysis and design

2001-11 Tom M. van Engers (VUA)
Knowledge Management:
The Role of Mental Models in Business Systems Design

====
2002
====

2002-01 Nico Lassing (VU)
Architecture-Level Modifiability Analysis

2002-02 Roelof van Zwol (UT)
Modelling and searching web-based document collections

2002-03 Henk Ernst Blok (UT)
Database Optimization Aspects for Information Retrieval

2002-04 Juan Roberto Castelo Valdueza (UU)
The Discrete Acyclic Digraph Markov Model in Data Mining

2002-05 Radu Serban (VU)
The Private Cyberspace Modeling Electronic Environments
inhabited by Privacy-concerned Agents

2002-06 Laurens Mommers (UL)
Applied legal epistemology;
Building a knowledge-based ontology of the legal domain

2002-07 Peter Boncz (CWI)
Monet: A Next-Generation DBMS Kernel For Query-Intensive Applications

2002-08 Jaap Gordijn (VU)
Value Based Requirements Engineering: Exploring Innovative
E-Commerce Ideas

2002-09 Willem-Jan van den Heuvel(KUB)
Integrating Modern Business Applications with Objectified Legacy Systems

2002-10 Brian Sheppard (UM)
Towards Perfect Play of Scrabble

2002-11 Wouter C.A. Wijngaards (VU)
Agent Based Modelling of Dynamics: Biological and Organisational Applications

2002-12 Albrecht Schmidt (Uva)
Processing XML in Database Systems

2002-13 Hongjing Wu (TUE)
A Reference Architecture for Adaptive Hypermedia Applications

2002-14 Wieke de Vries (UU)
Agent Interaction: Abstract Approaches to Modelling, Programming and
Verifying Multi-Agent Systems

2002-15 Rik Eshuis (UT)
Semantics and Verification of UML Activity Diagrams for Workflow Modelling

2002-16 Pieter van Langen (VU)
The Anatomy of Design: Foundations, Models and Applications

2002-17 Stefan Manegold (UVA)
Understanding, Modeling, and Improving Main-Memory Database Performance

====
2003
====

2003-01 Heiner Stuckenschmidt (VU)
Ontology-Based Information Sharing in Weakly Structured Environments

2003-02 Jan Broersen (VU)
Modal Action Logics for Reasoning About Reactive Systems

2003-03 Martijn Schuemie (TUD)
Human-Computer Interaction and Presence in Virtual Reality Exposure Therapy

2003-04 Milan Petkovic (UT)
Content-Based Video Retrieval Supported by Database Technology

2003-05 Jos Lehmann (UVA)
Causation in Artificial Intelligence and Law - A modelling approach

2003-06 Boris van Schooten (UT)
Development and specification of virtual environments

2003-07 Machiel Jansen (UvA)
Formal Explorations of Knowledge Intensive Tasks

2003-08 Yongping Ran (UM)
Repair Based Scheduling

2003-09 Rens Kortmann (UM)
The resolution of visually guided behaviour

2003-10 Andreas Lincke (UvT)
Electronic Business Negotiation: Some experimental studies on the interaction between medium, innovation context and culture

2003-11 Simon Keizer (UT)
Reasoning under Uncertainty in Natural Language Dialogue using Bayesian Networks

2003-12 Roeland Ordelman (UT)
Dutch speech recognition in multimedia information retrieval

2003-13 Jeroen Donkers (UM)
Nosce Hostem - Searching with Opponent Models

2003-14 Stijn Hoppenbrouwers (KUN)
Freezing Language: Conceptualisation Processes across ICT-Supported Organisations

2003-15 Mathijs de Weerdt (TUD)
Plan Merging in Multi-Agent Systems

2003-16 Menzo Windhouwer (CWI)
Feature Grammar Systems - Incremental Maintenance of Indexes to Digital Media Warehouses

2003-17 David Jansen (UT)
Extensions of Statecharts with Probability, Time, and Stochastic Timing

2003-18 Levente Kocsis (UM)
Learning Search Decisions

====
2004
====

2004-01 Virginia Dignum (UU)
A Model for Organizational Interaction: Based on Agents, Founded in Logic

2004-02 Lai Xu (UvT)
Monitoring Multi-party Contracts for E-business

2004-03 Perry Groot (VU)
A Theoretical and Empirical Analysis of Approximation in Symbolic Problem Solving

2004-04 Chris van Aart (UVA)
Organizational Principles for Multi-Agent Architectures

2004-05 Viara Popova (EUR)
Knowledge discovery and monotonicity

2004-06 Bart-Jan Hommes (TUD)
The Evaluation of Business Process Modeling Techniques

2004-07 Elise Boltjes (UM)
Voorbeeldig onderwijs; voorbeeldgestuurd onderwijs, een opstap naar abstract denken, vooral voor meisjes

2004-08 Joop Verbeek(UM)
Politie en de Nieuwe Internationale Informatiemarkt, Grensregionale politiële gegevensuitwisseling en digitale expertise

2004-09 Martin Caminada (VU)
For the Sake of the Argument; explorations into argument-based reasoning

2004-10 Suzanne Kabel (UVA)
Knowledge-rich indexing of learning-objects

2004-11 Michel Klein (VU)
Change Management for Distributed Ontologies

2004-12 The Duy Bui (UT)
Creating emotions and facial expressions for embodied agents

2004-13 Wojciech Jamroga (UT)
Using Multiple Models of Reality: On Agents who Know how to Play

2004-14 Paul Harrenstein (UU)
Logic in Conflict. Logical Explorations in Strategic Equilibrium

2004-15 Arno Knobbe (UU)
Multi-Relational Data Mining

2004-16 Federico Divina (VU)
Hybrid Genetic Relational Search for Inductive Learning

2004-17 Mark Winands (UM)
Informed Search in Complex Games

2004-18 Vania Bessa Machado (UvA)
Supporting the Construction of Qualitative Knowledge Models

2004-19 Thijs Westerveld (UT)
Using generative probabilistic models for multimedia retrieval

2004-20 Madelon Evers (Nyenrode)
Learning from Design: facilitating multidisciplinary design teams

====
2005

====

2005-01 Floor Verdenius (UVA)
Methodological Aspects of Designing Induction-Based Applications

2005-02 Erik van der Werf (UM))
AI techniques for the game of Go

2005-03 Franc Grootjen (RUN)
A Pragmatic Approach to the Conceptualisation of Language

2005-04 Nirvana Meratnia (UT)
Towards Database Support for Moving Object data

2005-05 Gabriel Infante-Lopez (UVA)
Two-Level Probabilistic Grammars for Natural Language Parsing

2005-06 Pieter Spronck (UM)
Adaptive Game AI

2005-07 Flavius Frasincar (TUE)
Hypermedia Presentation Generation for Semantic Web Information Systems

2005-08 Richard Vdovjak (TUE)
A Model-driven Approach for Building Distributed Ontology-based Web Applications

2005-09 Jeen Broekstra (VU)
Storage, Querying and Inferencing for Semantic Web Languages

2005-10 Anders Bouwer (UVA)
Explaining Behaviour: Using Qualitative Simulation in Interactive Learning Environments

2005-11 Elth Ogston (VU)
Agent Based Matchmaking and Clustering - A Decentralized Approach to Search

2005-12 Csaba Boer (EUR)
Distributed Simulation in Industry

2005-13 Fred Hamburg (UL)
Een Computermodel voor het Ondersteunen van Euthanasiebeslissingen

2005-14 Borys Omelayenko (VU)
Web-Service configuration on the Semantic Web; Exploring how semantics meets pragmatics

2005-15 Tibor Bosse (VU)
Analysis of the Dynamics of Cognitive Processes

2005-16 Joris Graaumans (UU)
Usability of XML Query Languages

2005-17 Boris Shishkov (TUD)
Software Specification Based on Re-usable Business Components

2005-18 Danielle Sent (UU)
Test-selection strategies for probabilistic networks

2005-19 Michel van Dartel (UM)
Situated Representation

2005-20 Cristina Coteanu (UL)
Cyber Consumer Law, State of the Art and Perspectives

2005-21 Wijnand Derks (UT)
Improving Concurrency and Recovery in Database Systems by
Exploiting Application Semantics

====
2006
====

2006-01 Samuil Angelov (TUE)
Foundations of B2B Electronic Contracting

2006-02 Cristina Chisalita (VU)
Contextual issues in the design and use of information technology in organizations

2006-03 Noor Christoph (UVA)
The role of metacognitive skills in learning to solve problems

2006-04 Marta Sabou (VU)
Building Web Service Ontologies

2006-05 Cees Pierik (UU)
Validation Techniques for Object-Oriented Proof Outlines

2006-06 Ziv Baida (VU)
Software-aided Service Bundling - Intelligent Methods & Tools
for Graphical Service Modeling

2006-07 Marko Smiljanic (UT)
XML schema matching – balancing efficiency and effectiveness by means of clustering

2006-08 Eelco Herder (UT)
Forward, Back and Home Again - Analyzing User Behavior on the Web

2006-09 Mohamed Wahdan (UM)
Automatic Formulation of the Auditor's Opinion

2006-10 Ronny Siebes (VU)
Semantic Routing in Peer-to-Peer Systems

2006-11 Joeri van Ruth (UT)
Flattening Queries over Nested Data Types

2006-12 Bert Bongers (VU)
Interactivation - Towards an e-cology of people, our technological environment, and the arts

2006-13 Henk-Jan Lebbink (UU)
Dialogue and Decision Games for Information Exchanging Agents

2006-14 Johan Hoorn (VU)
Software Requirements: Update, Upgrade, Redesign - towards a Theory of Requirements Change

2006-15 Rainer Malik (UU)
CONAN: Text Mining in the Biomedical Domain

2006-16 Carsten Riggelsen (UU)
Approximation Methods for Efficient Learning of Bayesian Networks

2006-17 Stacey Nagata (UU)
User Assistance for Multitasking with Interruptions on a Mobile Device

2006-18 Valentin Zhizhkun (UVA)
Graph transformation for Natural Language Processing

2006-19 Birna van Riemsdijk (UU)
Cognitive Agent Programming: A Semantic Approach

2006-20 Marina Velikova (UvT)
Monotone models for prediction in data mining

2006-21 Bas van Gils (RUN)
Aptness on the Web

2006-22 Paul de Vrieze (RUN)
Fundaments of Adaptive Personalisation

2006-23 Ion Juvina (UU)
Development of Cognitive Model for Navigating on the Web

2006-24 Laura Hollink (VU)
Semantic Annotation for Retrieval of Visual Resources

2006-25 Madalina Drugan (UU)
Conditional log-likelihood MDL and Evolutionary MCMC

2006-26 Vojkan Mihajlovic (UT)
Score Region Algebra: A Flexible Framework for Structured Information Retrieval

2006-27 Stefano Bocconi (CWI)
Vox Populi: generating video documentaries from semantically annotated media repositories

2006-28 Borkur Sigurbjornsson (UVA)
Focused Information Access using XML Element Retrieval

====
2007
====

2007-01 Kees Leune (UvT)
Access Control and Service-Oriented Architectures

2007-02 Wouter Teepe (RUG)
Reconciling Information Exchange and Confidentiality: A Formal Approach

2007-03 Peter Mika (VU)
Social Networks and the Semantic Web

2007-04 Jurriaan van Diggelen (UU)
Achieving Semantic Interoperability in Multi-agent Systems: a dialogue-based approach

2007-05 Bart Schermer (UL)
Software Agents, Surveillance, and the Right to Privacy:
a Legislative Framework for Agent-enabled Surveillance

2007-06 Gilad Mishne (UVA)
Applied Text Analytics for Blogs

2007-07 Natasa Jovanovic' (UT)
To Whom It May Concern - Addressee Identification in Face-to-Face Meetings

2007-08 Mark Hoogendoorn (VU)
Modeling of Change in Multi-Agent Organizations

2007-09 David Mobach (VU)
Agent-Based Mediated Service Negotiation

2007-10 Huib Aldewereld (UU)
Autonomy vs. Conformity: an Institutional Perspective on Norms and Protocols

2007-11 Natalia Stash (TUE)
Incorporating Cognitive/Learning Styles in a General-Purpose Adaptive Hypermedia System

2007-12 Marcel van Gerven (RUN)
Bayesian Networks for Clinical Decision Support:
A Rational Approach to Dynamic Decision-Making under Uncertainty

2007-13 Rutger Rienks (UT)
Meetings in Smart Environments; Implications of Progressing Technology

2007-14 Niek Bergboer (UM)
Context-Based Image Analysis

2007-15 Joyca Lacroix (UM)
NIM: a Situated Computational Memory Model

2007-16 Davide Grossi (UU)
Designing Invisible Handcuffs. Formal investigations in Institutions and Organizations for Multi-agent Systems

2007-17 Theodore Charitos (UU)
Reasoning with Dynamic Networks in Practice

2007-18 Bart Orriens (UvT)
On the development an management of adaptive business collaborations

2007-19 David Levy (UM)
Intimate relationships with artificial partners

2007-20 Slinger Jansen (UU)
Customer Configuration Updating in a Software Supply Network

2007-21 Karianne Vermaas (UU)
Fast diffusion and broadening use: A research on residential adoption and
usage of broadband internet in the Netherlands between 2001 and 2005

2007-22 Zlatko Zlatev (UT)
Goal-oriented design of value and process models from patterns

2007-23 Peter Barna (TUE)
Specification of Application Logic in Web Information Systems

2007-24 Georgina Ramrez Camps (CWI)
Structural Features in XML Retrieval

2007-25 Joost Schalken (VU)
Empirical Investigations in Software Process Improvement

====
2008
====

2008-01 Katalin Boer-Sorbn (EUR)
Agent-Based Simulation of Financial Markets: A modular,continuous-time approach

2008-02 Alexei Sharpanskykh (VU)
On Computer-Aided Methods for Modeling and Analysis of Organizations

2008-03 Vera Hollink (UVA)
Optimizing hierarchical menus: a usage-based approach

2008-04 Ander de Keijzer (UT)
Management of Uncertain Data - towards unattended integration

2008-05 Bela Mutschler (UT)
Modeling and simulating causal dependencies on process-aware information systems from a cost perspective

2008-06 Arjen Hommersom (RUN)
On the Application of Formal Methods to Clinical Guidelines, an Artificial Intelligence Perspective

2008-07 Peter van Rosmalen (OU)
Supporting the tutor in the design and support of adaptive e-learning

2008-08 Janneke Bolt (UU)
Bayesian Networks: Aspects of Approximate Inference

2008-09 Christof van Nimwegen (UU)
The paradox of the guided user: assistance can be counter-effective

2008-10 Wauter Bosma (UT)
Discourse oriented summarization

2008-11 Vera Kartseva (VU)
Designing Controls for Network Organizations: A Value-Based Approach

2008-12 Jozsef Farkas (RUN)
A Semiotically Oriented Cognitive Model of Knowledge Representation

2008-13 Caterina Carraciolo (UVA)
Topic Driven Access to Scientific Handbooks

2008-14 Arthur van Bunningen (UT)
Context-Aware Querying; Better Answers with Less Effort

2008-15 Martijn van Otterlo (UT)
The Logic of Adaptive Behavior: Knowledge Representation and Algorithms for the
Markov Decision Process Framework in First-Order Domains.

2008-16 Henriette van Vugt (VU)
Embodied agents from a user's perspective

2008-17 Martin Op 't Land (TUD)
Applying Architecture and Ontology to the Splitting and Allying of Enterprises

2008-18 Guido de Croon (UM)
Adaptive Active Vision

2008-19 Henning Rode (UT)
From Document to Entity Retrieval: Improving Precision and Performance of Focused Text Search

2008-20 Rex Arendsen (UVA)
Geen bericht, goed bericht. Een onderzoek naar de effecten van de introductie
van elektronisch berichtenverkeer met de overheid op de administratieve lasten van bedrijven

2008-21 Krisztian Balog (UVA)
People Search in the Enterprise

2008-22 Henk Koning (UU)
Communication of IT-Architecture

2008-23 Stefan Visscher (UU)
Bayesian network models for the management of ventilator-associated pneumonia

2008-24 Zharko Aleksovski (VU)
Using background knowledge in ontology matching

2008-25 Geert Jonker (UU)
Efficient and Equitable Exchange in Air Traffic Management Plan Repair using Spender-signed Currency

2008-26 Marijn Huijbregts (UT)
Segmentation, Diarization and Speech Transcription: Surprise Data Unraveled

2008-27 Hubert Vogten (OU)
Design and Implementation Strategies for IMS Learning Design

2008-28 Ildiko Flesch (RUN)
On the Use of Independence Relations in Bayesian Networks

2008-29 Dennis Reidsma (UT)

Annotations and Subjective Machines - Of Annotators, Embodied Agents, Users, and Other Humans

2008-30 Wouter van Atteveldt (VU)
Semantic Network Analysis: Techniques for Extracting, Representing and Querying Media Content

2008-31 Loes Braun (UM)
Pro-Active Medical Information Retrieval

2008-32 Trung H. Bui (UT)
Toward Affective Dialogue Management using Partially Observable Markov Decision Processes

2008-33 Frank Terpstra (UVA)
Scientific Workflow Design; theoretical and practical issues

2008-34 Jeroen de Knijf (UU)
Studies in Frequent Tree Mining

2008-35 Ben Torben Nielsen (UvT)
Dendritic morphologies: function shapes structure

====
2009
====

2009-01 Rasa Jurgelenaite (RUN)
Symmetric Causal Independence Models

2009-02 Willem Robert van Hage (VU)
Evaluating Ontology-Alignment Techniques

2009-03 Hans Stol (UvT)
A Framework for Evidence-based Policy Making Using IT

2009-04 Josephine Nabukenya (RUN)
Improving the Quality of Organisational Policy Making using Collaboration Engineering

2009-05 Sietse Overbeek (RUN)
Bridging Supply and Demand for Knowledge Intensive Tasks - Based on Knowledge, Cognition, and Quality

2009-06 Muhammad Subianto (UU)
Understanding Classification

2009-07 Ronald Poppe (UT)
Discriminative Vision-Based Recovery and Recognition of Human Motion

2009-08 Volker Nannen (VU)
Evolutionary Agent-Based Policy Analysis in Dynamic Environments

2009-09 Benjamin Kanagwa (RUN)
Design, Discovery and Construction of Service-oriented Systems

2009-10 Jan Wielemaker (UVA)
Logic programming for knowledge-intensive interactive applications

2009-11 Alexander Boer (UVA)
Legal Theory, Sources of Law & the Semantic Web

2009-12 Peter Massuthe (TUE, Humboldt-Universitaet zu Berlin)
perating Guidelines for Services

2009-13 Steven de Jong (UM)
Fairness in Multi-Agent Systems

2009-14 Maksym Korotkiy (VU)

From ontology-enabled services to service-enabled ontologies
(making ontologies work in e-science with ONTO-SOA)

2009-15 Rinke Hoekstra (UVA)
Ontology Representation - Design Patterns and Ontologies that Make Sense
2009-16 Fritz Reul (UvT)
New Architectures in Computer Chess
2009-17 Laurens van der Maaten (UvT)
Feature Extraction from Visual Data
2009-18 Fabian Groffen (CWI)
Armada, An Evolving Database System

2009-19 Valentin Robu (CWI)
Modeling Preferences, Strategic Reasoning and Collaboration in Agent-Mediated Electronic Markets

2009-20 Bob van der Vecht (UU)
Adjustable Autonomy: Controling Influences on Decision Making

2009-21 Stijn Vanderlooy (UM)
Ranking and Reliable Classification

2009-22 Pavel Serdyukov (UT)
Search For Expertise: Going beyond direct evidence

2009-23 Peter Hofgesang (VU)
Modelling Web Usage in a Changing Environment

2009-24 Annerieke Heuvelink (VUA)
Cognitive Models for Training Simulations

2009-25 Alex van Ballegooij (CWI)
"RAM: Array Database Management through Relational Mapping"

2009-26 Fernando Koch (UU)
An Agent-Based Model for the Development of Intelligent Mobile Services

2009-27 Christian Glahn (OU)
Contextual Support of social Engagement and Reflection on the Web

2009-28 Sander Evers (UT)
Sensor Data Management with Probabilistic Models

2009-29 Stanislav Pokraev (UT)
Model-Driven Semantic Integration of Service-Oriented Applications

2009-30 Marcin Zukowski (CWI)
Balancing vectorized query execution with bandwidth-optimized storage

2009-31 Sofiya Katrenko (UVA)
A Closer Look at Learning Relations from Text

2009-32 Rik Farenhorst (VU) and Remco de Boer (VU)
Architectural Knowledge Management: Supporting Architects and Auditors

2009-33 Khiet Truong (UT)
How Does Real Affect Affect Affect Recognition In Speech?

2009-34 Inge van de Weerd (UU)
Advancing in Software Product Management: An Incremental Method Engineering Approach

2009-35 Wouter Koelewijn (UL)
Privacy en Politiegegevens; Over geautomatiseerde normatieve informatie-uitwisseling

2009-36 Marco Kalz (OUN)
Placement Support for Learners in Learning Networks

2009-37 Hendrik Drachsler (OUN)

Navigation Support for Learners in Informal Learning Networks

2009-38 Riina Vuorikari (OU)
Tags and self-organisation: a metadata ecology for learning resources in a multilingual context

2009-39 Christian Stahl (TUE, Humboldt-Universitaet zu Berlin)
Service Substitution – A Behavioral Approach Based on Petri Nets

2009-40 Stephan Raaijmakers (UvT)
Multinomial Language Learning: Investigations into the Geometry of Language

2009-41 Igor Berezhnyy (UvT)
Digital Analysis of Paintings

2009-42 Toine Bogers
Recommender Systems for Social Bookmarking

2009-43 Virginia Nunes Leal Franqueira (UT)
Finding Multi-step Attacks in Computer Networks using Heuristic Search and Mobile Ambients

2009-44 Roberto Santana Tapia (UT)
Assessing Business-IT Alignment in Networked Organizations

2009-45 Jilles Vreeken (UU)
Making Pattern Mining Useful

2009-46 Loredana Afanasiev (UvA)
Querying XML: Benchmarks and Recursion

====
2009
====

2010-01 Matthijs van Leeuwen (UU)
Patterns that Matter

2010-02 Ingo Wassink (UT)
Work flows in Life Science

2010-03 Joost Geurts (CWI)
A Document Engineering Model and Processing Framework for Multimedia documents
2010-04 Olga Kulyk (UT)
Do You Know What I Know? Situational Awareness of Co-located Teams in Multidisplay Environments

2010-05 Claudia Hauff (UT)
Predicting the Effectiveness of Queries and Retrieval Systems

2010-06 Sander Bakkes (UvT)
Rapid Adaptation of Video Game AI

2010-07 Wim Fikkert (UT)
Gesture interaction at a Distance

2010-08 Krzysztof Siewicz (UL)
Towards an Improved Regulatory Framework of Free Software. Protecting user freedoms in a world of software communities and eGovernments

2010-09 Hugo Kielman (UL)
A Politiele gegevensverwerking en Privacy, Naar een effectieve waarborging

2010-10 Rebecca Ong (UL)
Mobile Communication and Protection of Children 2010-11 Adriaan Ter Mors (TUD)
The world according to MARP: Multi-Agent Route Planning

2010-12 Susan van den Braak (UU)
Sensemaking software for crime analysis

2010-13 Gianluigi Folino (RUN)
High Performance Data Mining using Bio-inspired techniques

2010-14 Sander van Splunter (VU)
Automated Web Service Reconfiguration

2010-15 Lianne Bodenstaff (UT)
Managing Dependency Relations in Inter-Organizational Models

2010-16 Sicco Verwer (TUD)
Efficient Identification of Timed Automata, theory and practice

2010-17 Spyros Kotoulas (VU)
Scalable Discovery of Networked Resources: Algorithms, Infrastructure, Applications

2010-18 Charlotte Gerritsen (VU)
Caught in the Act: Investigating Crime by Agent-Based Simulation

2010-19 Henriette Cramer (UvA)
People's Responses to Autonomous and Adaptive Systems

2010-20 Ivo Swartjes (UT)
Whose Story Is It Anyway? How Improv Informs Agency and Authorship of Emergent Narrative

2010-21 Harold van Heerde (UT)
Privacy-aware data management by means of data degradation

2010-22 Michiel Hildebrand (CWI)
End-user Support for Access to Heterogeneous Linked Data

2010-23 Bas Steunebrink (UU)
The Logical Structure of Emotions

2010-24 Dmytro Tykhonov
Designing Generic and Efficient Negotiation Strategies

2010-25 Zulfiqar Ali Memon (VU)
Modelling Human-Awareness for Ambient Agents: A Human Mindreading Perspective

2010-26 Ying Zhang (CWI)
XRPC: Efficient Distributed Query Processing on Heterogeneous XQuery Engines

2010-27 Marten Voulon (UL)
Automatisch contracteren

2010-28 Arne Koopman (UU)
Characteristic Relational Patterns

2010-29 Stratos Idreos(CWI)
Database Cracking: Towards Auto-tuning Database Kernels

2010-30 Marieke van Erp (UvT)
Accessing Natural History - Discoveries in data cleaning, structuring, and retrieval

2010-31 Victor de Boer (UVA)
Ontology Enrichment from Heterogeneous Sources on the Web

2010-32 Marcel Hiel (UvT)
An Adaptive Service Oriented Architecture: Automatically solving Interoperability Problems

2010-33 Robin Aly (UT)
Modeling Representation Uncertainty in Concept-Based Multimedia Retrieval

2010-34 Teduh Dirgahayu (UT)
Interaction Design in Service Compositions

2010-35 Dolf Trieschnigg (UT)
Proof of Concept: Concept-based Biomedical Information Retrieval

2010-36 Jose Janssen (OU)
Paving the Way for Lifelong Learning;
Facilitating competence development through a learning path specification

2010-37 Niels Lohmann (TUE)
Correctness of services and their composition

2010-38 Dirk Fahland (TUE)
From Scenarios to components

2010-39 Ghazanfar Farooq Siddiqui (VU)
Integrative modeling of emotions in virtual agents

2010-40 Mark van Assem (VU)
Converting and Integrating Vocabularies for the Semantic Web

2010-41 Guillaume Chaslot (UM)
Monte-Carlo Tree Search

2010-42 Sybren de Kinderen (VU)
Needs-driven service bundling in a multi-supplier setting - the computational e3-service approach

2010-43 Peter van Kranenburg (UU)
A Computational Approach to Content-Based Retrieval of Folk Song Melodies

2010-44 Pieter Bellekens (TUE)
An Approach towards Context-sensitive and User-adapted Access to
Heterogeneous Data Sources, Illustrated in the Television Domain

2010-45 Vasilios Andrikopoulos (UvT)
A theory and model for the evolution of software services

2010-46 Vincent Pijpers (VU)
e3alignment: Exploring Inter-Organizational Business-ICT Alignment

2010-47 Chen Li (UT)
Mining Process Model Variants: Challenges, Techniques, Examples

During the last years a new generation of process-aware information systems has emerged, which enables process model configurations at buildtime as well as process instance ...

ISBN: 978-90-365-3084-2