Flexible evolutionary algorithms for mining structured process models

DOI:
10.6100/IR780920

Flexible Evolutionary Algorithms for Mining Structured Process Models

J.C.A.M. Buijs

CIP-DATA LIBRARY TECHNISCHE UNIVERSITEIT EINDHOVEN

Buijs, J.C.A.M.

Flexible Evolutionary Algorithms for Mining Structured Process Models
/ by J.C.A.M. Buijs.
Eindhoven: Technische Universiteit Eindhoven, 2014. Proefschrift.

A catalogue record is available from the Eindhoven University of Technology Library

ISBN 978-90-386-3681-8

Keywords: business process management, evolutionary algorithm, process mining

The work in this thesis has been sponsored by JACQUARD / NWO under the CoSeLoG project (638.001.211).

SIKS Dissertation Series No. 2014-36
The research reported in this thesis has been carried out under the auspices of SIKS, the Dutch Research School for Information and Knowledge Systems.

Cover design and printing by proefschriftmaken.nl

Flexible Evolutionary Algorithms for Mining Structured Process Models

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties, in het openbaar te verdedigen op
dinsdag 28 oktober 2014 om 16:00 uur

door

Joseph Cornelis Antonius Maria Buijs

geboren te Breda

Dit proefschrift is goedgekeurd door de promotoren en de samenstelling van de promotiecommissie is als volgt:

voorzitter:	prof.dr. E.H.L. Aarts
1e promotor:	prof.dr.ir. W.M.P. van der Aalst
co-promotor:	dr.ir. B.F. van Dongen
externe leden:	prof.dr.ir. A.H.M. ter Hofstede (Queensland University of Technology)
	prof.dr. M. Dumas (University of Tartu)
overige leden:	prof.dr.ir. H.A. Reijers
	prof.dr. B. Hammer (Bielefeld University)
lid TU/e:	prof.dr.ir. J.J. van Wijk

Dedicated to my family.

Abstract

Process mining automatically produces a process model while considering only an organization's records of its operational processes. Over the last decade, many process discovery techniques have been developed, and many authors have compared these techniques by focusing on the properties of the models produced. However, none of the current techniques guarantee to produce sound (i.e., syntactically correct) process models. Furthermore, none of the current techniques provide insights into the trade-offs between the different quality dimensions of process models.

In this thesis we present the Evolutionary Tree Miner (ETM) framework. Its main feature is the guarantee that the discovered process models are sound. Another feature is that the ETM framework also incorporates all four well-known quality dimensions in process discovery (replay fitness, precision, generalization and simplicity). Additional quality metrics can be easily added to the Evolutionary Tree Miner. The Evolutionary Tree Miner framework is able to balance these different quality metrics and is able to produce (a collection of) process models that have a specific balance of these quality dimensions, as specified by the user.

The third main feature of the Evolutionary Tree Miner is that it is easily extensible. In this thesis we discuss extensions for the discovery of a collection of process models with different quality trade-offs, the discovery of (a collection of) process models using a given process model, and the discovery of a configurable process model that describes multiple event-logs.

The Evolutionary Tree Miner is implemented as a plug-in for the process mining toolkit ProM. The Evolutionary Tree Miner and all of its extensions are evaluated using both artificial and real-life data sets.

Contents

Abstract **vii**

List of Figures **xvii**

List of Tables **xxv**

1 Introduction **1**

1.1 Process Mining 2
1.2 Introduction of Running Examples 5
1.2.1 System with two Event Logs 6
1.2.2 Four Similar Processes 7
1.3 Challenges in Process Mining 10
1.3.1 Results of Existing Process Discovery Techniques 10
1.3.2 Challenge 1: Produce Correct Process Models 11
1.3.3 Challenge 2: Separation of Visualization and Representational Bias 11
1.3.4 Challenge 3: Balance the Quality of Discovered Process Models 13
1.3.5 Challenge 4: Improve Understandability for Non-Experts 14
1.3.6 Challenge 5: Use Existing Knowledge in Process Discovery 15
1.3.7 Challenge 6: Describe a Family of Processes 16
1.3.8 Challenge 7: Compare Similar Observed Behavior 17
1.3.9 An Algorithm that Addresses all Challenges 17
1.4 The CoSeLoG Project 18

1.5 Contributions and Structure of this Thesis 22

2 Preliminaries 25
2.1 Notations 25
2.2 Process Models 27
2.2.1 Labeled Transition Systems 27
2.2.2 Petri Nets 28
2.2.3 Business Process Model and Notation (BPMN) 31
2.3 Event Logs 32

3 Process Trees 35
3.1 Requirements 35
3.1.1 Soundness and Relaxed Soundness 36
3.1.2 Expressiveness 37
3.1.3 Understandability 39
3.1.4 Formal Semantics 39
3.1.5 Suitable for the Process Discovery Algorithm 40
3.2 Common Process Modeling Notations 40
3.2.1 Hidden Markov Models 40
3.2.2 Yet Another Workflow Language (YAWL) 41
3.2.3 Event-Driven Process Chains (EPCs) 43
3.2.4 Causal Nets 43
3.2.5 Heuristics Net 45
3.2.6 Fuzzy Models 46
3.2.7 Process Algebras 46
3.3 Notations v.s. Requirements 48
3.4 The Process Tree Notation 50
3.5 Translations 57
3.5.1 From Process Trees to Other Notations 57
3.5.2 From other Notations to Process Trees 68
3.6 Conclusion 73

4 A Framework for Evolutionary Process Mining 75
4.1 The ETM Framework 75
4.2 Applications of the Evolutionary Framework 77
4.2.1 Process Discovery 78
4.2.2 Process Model Repair 79
4.2.3 Process Discovery of a Configurable Model 79
4.2.4 Configuration Discovery using Context 81

4.2.5 Concept Drift 81
4.2.6 Decision Mining 83
4.2.7 Other Perspectives 83
4.2.8 Combinations of Scenarios 85
4.3 General Requirements for Evolutionary Algorithms 85
4.3.1 Population Diversity 86
4.3.2 Ability to Visit the Whole Search Space 86
4.3.3 Prevention of Bloat 87
4.3.4 Requirements for the Evaluation of Candidates 87
4.4 Common Implementations of the Phases of an Evolutionary Algorithm 89
4.4.1 Candidate Evaluation 89
4.4.2 Selection 91
4.4.3 Change Operations 93
4.4.4 Termination 94
4.5 Conclusion 95

5 Process Model Quality Dimensions 97
5.1 The Four Process Discovery Quality Dimensions 98
5.2 Theoretical View 99
5.2.1 Relating the Behavior of the Event Log, the Process Model and the System 100
5.2.2 Dealing with an Unknown System 102
5.3 Simplicity 105
5.3.1 Simplicity by Ratio of Useless Nodes 105
5.3.2 Other Simplicity Metrics 107
5.4 Replay Fitness 108
5.4.1 Alignment-based Replay Fitness 108
5.4.2 Other Replay Fitness Metrics 111
5.5 Precision 113
5.5.1 Escaping Edges 114
5.5.2 Other Precision Metrics 115
5.6 Generalization 116
5.6.1 Frequency of Use 117
5.6.2 Other Generalization Metrics 118
5.7 The Importance of Considering all Four Quality Dimensions . . . 120
5.8 Quality Metric Considerations 120
5.9 Additional Quality Dimensions 122
5.10 Related Work 122

5.11 Conclusion 123

6 Discovery of Process Trees 125

6.1 Initial Population Creation 125
6.1.1 Random Tree Creation 127
6.1.2 Using Advanced Trace-Model Creation 127
6.1.3 Using Other Process Discovery Algorithms 129
6.2 Random Mutation 131
6.3 Guided Mutation 134
6.3.1 Removing Behavior 134
6.3.2 Adding Behavior 135
6.3.3 Changing Behavior 136
6.4 Crossover 137
6.5 Candidate Selection 138
6.6 Termination Conditions 139
6.7 Balancing Search Space Exploration and Exploitation 140
6.8 Application Using a Running Example 142
6.8.1 Searching for the Best Process Tree 142
6.8.2 Discovery of a Pareto Front 145
6.9 Results of Existing Process Discovery Algorithms 145
6.9.1 The α-Algorithm 147
6.9.2 Genetic Miner 148
6.9.3 Heuristic Miner 149
6.9.4 The ILP Miner 151
6.9.5 Inductive Miner 152
6.9.6 Language-based Region Theory 153
6.9.7 Multi-phase Miner 156
6.9.8 State-based Region Theory 157
6.9.9 Why Existing Algorithms Fail 159
6.10 Conclusion 160

7 Application of Process Tree Discovery 163

7.1 Performance in the Limit 163
7.1.1 Running Example without Exceptional Behavior 164
7.1.2 Running Example with Exceptional Behavior 170
7.1.3 Pareto Front Evolution on Running Example with Exceptional Behavior 174
7.2 Random versus Guided Change 183
7.3 Building Permits Process - Receipt Phase 186

7.4 Building Permits Process - Objections and Complaints 192
7.5 Performance of the ETM*d* algorithm 197
7.6 Conclusion 201

8 Balancing Observed and Modeled Behavior 203
8.1 Application Scenarios 204
8.2 Similarity as the 5th Quality Dimension 205
8.3 Application to Running Example 208
8.4 Case Study 214
8.5 Related Work 221
8.6 Conclusion 222

9 Discovering Configurable Process Models 223
9.1 Configurable Process Models 224
9.2 Configurable Process Trees 228
9.3 Four Different Approaches 230
9.4 The ETM*c* algorithm 232
9.4.1 Configuration Mutation 233
9.4.2 Configuration Quality 233
9.5 Application on Running Example 234
9.5.1 Experimental Setup 234
9.5.2 Approach 1: Merge Individually Discovered Process Models 235
9.5.3 Approach 2: Merge Similar Discovered Process Models . . 237
9.5.4 Approach 3: First Discover a Single Process Model and Then Discover Configurations 240
9.5.5 Approach 4: Discover Process Model and Configurations at the Same Time 241
9.5.6 Discovering a Pareto Front for Approach 4 242
9.5.7 Comparison of the Four Approaches 247
9.6 Case Study 247
9.7 Related Work 257
9.8 Conclusion 258

10 Inter-Organizational Process Comparison 259
10.1 Running Example 260
10.2 Cross-Organizational Comparison Framework 261
10.2.1 Process Model Metrics 263
10.2.2 Event Log Metrics 263

10.2.3 Comparison Metrics 263
10.2.4 Application on the Running Example 264
10.3 Visualizing Alignments: the Alignment Matrix 265
10.4 Case Study 268
10.4.1 Setup 268
10.4.2 Execution 269
10.4.3 Results 273
10.5 Related Work 274
10.6 Conclusion 275

11 Implementation 277
11.1 Walk through of the ETM*d* Algorithm 277
11.1.1 Usage via the GUI 278
11.1.2 Usage via code 285
11.1.3 Usage via Command Line Interface 285
11.2 Extending the ETM*d* Algorithm 288
11.2.1 Adding Quality Metrics 288
11.2.2 Change Operations and Process Tree Creation 289
11.2.3 Pareto Front Visualizers 290
11.3 Implementation of the Comparison Framework 290
11.3.1 Metric Settings 290
11.3.2 Alignment Matrix Settings 293
11.3.3 Extending the Comparison Framework 294
11.4 Conclusion 295

12 Conclusion 297
12.1 Contributions of this Thesis 297
12.2 Current Challenges and Open Issues 300
12.2.1 Limitations 300
12.2.2 Improvements 301
12.2.3 Opportunities 303
12.3 Outlook on Process Mining 305
12.3.1 Closer Collaboration between Academia, Tool Vendors, Consultants and Clients 306
12.3.2 Address Hindering Side Issues 306
12.3.3 Improve the Applicability of Solutions on Real Data 307
12.3.4 Improve the Usability of Techniques 308
12.3.5 Create Incentives for Researchers to Publish and Document their Solutions 308

Bibliography 311

Index 335

Summary 339

Samenvatting 341

Acknowledgments 343

Curriculum Vitae 347

SIKS dissertations 351

List of Figures

1.1 Positioning of Process Mining. 3
1.2 The four quality dimensions used to qualify a process model given the observed behavior. 5
1.3 Process model as executed in the system, represented by a Petri net. 6
1.4 Petri net process models for the four loan application process variants. 9
1.5 Result of the heuristics miner on the running example event log. 12
1.6 Two example process models, discovered from the running examples without and with exceptional behavior. 13
1.7 Alternative process model for the running example. 14
1.8 Configurable process model for variants 1 and 3 of Figure 1.4. . . 17
1.9 Traditional situation and envisioned situation within the CoSeLoG project. 19

2.1 A labeled transition system with 13 states and 21 transitions. . . . 28
2.2 Example of a Petri net. 29
2.3 Example of a BPMN model. 32

3.1 Examples of unsound Petri nets. 37
3.2 An example HMM model, with 11 (hidden) states and 11 possible observations. 41
3.3 An example YAWL process model where g cancels a region. . . . 42
3.4 An example EPC model. 44

3.5 An example causal net. 45
3.6 An example of a Heuristics net. 46
3.7 Example of Fuzzy models describing the same behavior as the previous models. 47
3.8 Example process tree and its Petri net translation. 52
3.9 Example process tree with for each node the associated index. . . 54
3.10 Process model that can easily be made block-structured, and its translation to a process tree. 69
3.11 Process model that is not block-structured but sound, and its translation to a process tree. 70
3.12 Process model that is not block-structured and unsound, and its translation to a process tree. 71
3.13 Example CoSeNet from [161] and the corresponding Process Tree translation. 72

4.1 The basic framework for evolutionary process discovery. 76
4.2 Process discovery scenario. 78
4.3 Process model repair scenario. 79
4.4 Scenario of the discovery of a configurable process model and its configurations. 80
4.5 Scenario of the discovery of a configurable process model, its configurations and the context of the log splitting. 81
4.6 Scenario of the discovery of a configurable process model, its configurations and the time information of the log splitting. . . . 82
4.7 Decision mining scenario. 83
4.8 Pareto Front of the two quality dimensions replay fitness and precision. 91

5.1 The four quality dimensions for process models in process discovery. 98
5.2 Venn diagram showing that the behavior of the process model (M), event log (L) and system (S) can be disjoint or overlapping. 99
5.3 Running example used to explain the quality dimensions in process discovery. 104
5.4 Process tree with several useless nodes. 107
5.5 Process model without perfect replay fitness. 108
5.6 Example of a token-based replay result. 112
5.7 Example process model to explain precision 114
5.8 Partial state space as constructed during alignment calculation. . 114

5.9 Example process model which is not general. 116
5.10 Four process models with different quality scores. 121

6.1 The basic framework for evolutionary process discovery. 126
6.2 Example of random construction of a process tree. 127
6.3 Example of advanced trace model creation. 128
6.4 Merging of trace models. 130
6.5 Three types of node addition mutation where leaf y is added. . . 132
6.6 Removing behavior. 134
6.7 Adding behavior. 135
6.8 Changing behavior. 136
6.9 Example of crossover applied on two parents, creating two offspring. 138
6.10 Running example used to discover a process tree (see Section 1.2).143
6.11 Results of the ETM*d* algorithm on the running example event logs. 144
6.12 Examples of process trees with slightly different trade-offs between the quality dimensions. 146
6.13 Results of the α-algorithm [20] on the running examples. 147
6.14 Result of the genetic miner [29] (Relaxed sound, tokens left behind). 148
6.15 Result of the heuristic miner [180] (relaxed sound since tokens are left behind). 150
6.16 Result of the ILP miner [182] (Ensuring empty net after completion). 151
6.17 Results of the Inductive Miner. 153
6.18 Result of the language-based region theory. 155
6.19 Result of the Multi-phase miner [66]. 156
6.20 Result of the state-based region theory. 158

7.1 Running example used to discover a process tree. 165
7.2 Minimum, average and maximum overall quality of the best candidates over 30 runs of ETM*d* on the running example, all 10,000 runs. 166
7.3 Minimum, average and maximum overall quality of the best candidates over 30 runs of ETM*d* on the running example, first 100 generations. 167
7.4 Process trees discovered by the ETM*d* algorithm on the running example in different generations. 168
7.5 Results of the ETM*d* on the running examples. 169

7.6 Minimum, average and maximum overall quality of the best candidates over 30 runs of ETM*d* on the running example with exceptional behavior, all 10,000 generations. 170
7.7 Minimum, average and maximum overall quality of the best candidates over 30 runs of ETM*d* on the running example with exceptional behavior, the first 100 generations. 171
7.8 Process trees discovered by the ETM*d* algorithm on the running example with exceptional behavior in different generations. . . . 173
7.9 Minimum, average and maximum size of the Pareto front over 30 runs of the ETM*d* algorithm on the running example with exceptional behavior, all 10,000 generations. 175
7.10 Distribution of the candidates over the quality dimensions in the (unprocessed) Pareto front of 1,996 candidates. 176
7.11 Distribution of the candidates over the quality dimensions in the normalized Pareto front containing 74 candidates. 177
7.12 Example of process tree before (a) and after (b) post-processing which normalizes a process tree by removing useless nodes. . . . 179
7.13 Visualization of the candidates in the normalized Pareto front. . 180
7.14 Visualization of the candidates in the normalized Pareto front with both replay fitness and precision above 0.9. 181
7.15 Examples of process trees from the Pareto front of Figure 7.13 with slightly different trade-offs between the quality dimensions. 182
7.16 Comparison of different ratios of random mutation on the running example with exceptional behavior. 184
7.17 Comparison of different ratios of random change operators on the building permits event log over 1,000 generations. 185
7.18 Visualization of the candidates in the Pareto front discovered for the building permits event log. 188
7.19 Process trees from the Pareto front for the building permits event log. 190
7.20 Results of the α-algorithm, Heuristics miner and Inductive Miner on the 'building permits - receipt phase' event log. 191
7.21 Visualization of the candidates in the Pareto front discovered for the WABO event logs. 194
7.22 Process trees for each of the event logs with a good balance of replay fitness and precision with high generalization (other event logs follow on next page). 195

7.22 (continued) Process trees for each of the event logs with a good balance between replay fitness and precision with high generalization (WABO2_BB on previous page). 196
7.23 Performance statistics per generation for the running example. . 198
7.24 Performance statistics per generation while constructing a Pareto front. 198
7.25 Performance statistics per generation for the WABO event logs. . 200

8.1 Similarity as the 5th quality dimension, influencing the other 4 quality dimensions. 205
8.2 Examples of possible edits on a tree (a) and respective similarities. 206
8.3 Process tree as known within the company. 208
8.4 Distribution of the candidates over the quality dimensions in the Pareto front of 573 candidates for the running example. 210
8.5 Visualization of the Pareto front of process trees discovered by the ETM*r* algorithm on the running example event log. 211
8.6 Visualizations of the Pareto front discovered by the ETM*r* algorithm on the running example, filtered by the number of edits allowed. 212
8.7 Process trees discovered while maintaining similarity with the normative model for the running example. 213
8.8 Process tree used as the normative model for the case study application of the ETM*r* algorithm. 214
8.9 Distribution over the quality dimensions of 200 candidates in the Pareto front for the case study. 215
8.10 Visualization of the Pareto front of process trees discovered by the ETM*r* algorithm on the case study event log. 216
8.11 Visualization of the Pareto front filtered by the number of edits allowed. 219
8.12 Different process trees selected from the Pareto front discovered by the ETM*r* algorithm on the case study. 220

9.1 Example of a configurable process model in C-EPC notation. . . . 225
9.2 Examples of configurable process models in C-YAWL notation. . . 227
9.3 Example of a configurable process model in PROVOP notation. . 227
9.4 Effects of blocking nodes in a process tree 229
9.5 Hierarchy of operator downgrade options. 230
9.6 Example of downgrading an ∨-operator to an →-operator. 230

9.7 Four approaches to creating a configurable process model from a collection of event logs. 231
9.8 Results of Approach 1, merging separately discovered process models, on the running example. 236
9.9 Individual results of approach 2, merging the similar process models, on the running example. 238
9.10 Configurable process model with quality statistics as discovered by Approach 2, merging the similar process models, on the running example. 239
9.11 Results of Approach 3, the two-phase mining approach, on the running example. 240
9.12 Results of Approach 4, the integrated mining approach, on the running example. 241
9.13 Distribution of the 382 candidates in the Pareto front over the quality dimensions for the running example. 243
9.14 Visualization of the Pareto front of configurable process trees discovered by the ETMc algorithm on the running example event logs. 244
9.15 Configurable process tree in the Pareto front discovered by Approach 4 for the running example event logs that balances replay fitness and precision. 245
9.16 Configurable process tree in the Pareto front discovered by Approach 4 for the running example event logs that has good replay fitness at the cost of precision. 246
9.17 Results of Approach 1, merging separate discovered process models, on the case study event logs. 248
9.18 Results of Approach 2, merging the similar process models, on the case study event logs. 249
9.19 Results of Approach 3, the two-phase mining approach, on the case study event logs. 250
9.20 Results of Approach 4, the integrated mining approach, on the case study event logs. 251
9.21 Distribution of the 44 candidates in the Pareto front for the case study over the quality dimensions. 253
9.22 Visualization of the Pareto front of configurable process trees discovered by the ETMc algorithm on the case study event logs. . . 254
9.23 Configurable process tree found by Approach 4 for the case study event logs that has the best overall replay fitness (0.972). 255

9.24 Configurable process tree found by Approach 4 for the case study event logs with best trade-off between replay fitness and precision.256

10.1 The process trees of the four variants of the running example (see Section 1.2.2). 261
10.2 Alignments and the construction of the alignment matrix. 265
10.3 Two of the analysis results shown to the case study participants. 270
10.4 A photo and an annotated alignment matrix with some observations made during the case study. 272

11.1 Plug-ins provided by the ETM package in ProM. 279
11.2 ETM parameter wizard screens. 281
11.3 Result of the 'Mine Pareto front with ETM*d* in Live mode'-plugin. 282
11.4 Different visualizations and navigators for the live Pareto front visualization. 284
11.5 Main user interface of the comparison framework. 291
11.6 Overview of the main, event log and model statistic settings panel as well as the LaTeX export settings panel. 292
11.7 Overview of the alignment matrix comparison statistic, and its settings for the column, row and cell representation. 294

12.1 Example of a 'long-term dependency' (LTD) operator, which synchronizes the choices of the two ×-operators. 303

List of Tables

1.1 Example event data. 4

1.2 Running example event logs, with and without exceptional behavior. 7

1.3 Event logs of the four loan application process variants. 8

1.4 Comparison of results of process discovery algorithms on the running example event logs. 10

1.5 Example of a way to compare the four process variants. 18

2.1 Example event data. 33

3.1 Classification of common process modeling notations on process modeling notation requirements. 49

3.2 Process tree operators, their Petri net translation and their allowed traces. 59

3.3 Process tree operators, their translations to YAWL and examples of allowed traces. 61

3.4 Process tree operators and their translations to BPMN parts. . . . 63

3.5 Process tree operators and their EPC translations. 65

3.6 Process tree operators and their CCS translation. 67

5.1 Overview of the different fractions and quality dimensions. . . . 103

5.2 Optimal alignments of the three traces of Figure 5.3b on the process model of Figure 5.5. 110

6.1 Overview of the different operators defined and their contribution to the search. 141
6.2 Quality of results of process discovery algorithms on the running example event log. 159
6.3 Quality of results of process discovery algorithms on the running example event log. 160

7.1 Statistics of the event logs used in this chapter. 164
7.2 Activity codes and corresponding names for the building permits - receipt phase event log. 187
7.3 Activity codes and corresponding names for the 'WABO' event logs.193

8.1 The event log 208

9.1 Four event logs for the four different variants of the loan application process. 234
9.2 Case study event log statistics 248

10.1 Four event logs for the four different variants of the loan application process of Figure 10.1. 262
10.2 The comparison table of the comparison framework approach. . 262
10.3 Application of the comparison framework on the running example 264
10.4 Application of the alignment matrices on the running example. . 267

Chapter 1

Introduction

Nowadays most organizations use information systems to support the execution of their business processes [72]. These information systems guide and support the execution of these processes by storing and sharing information, and by distributing activities and messages between participants.

More and more organizations distribute the execution of a process over different locations. Examples include multinational organizations such as Shell, Hertz, Unilever, Ahold and Philips. It could be that a particular business process, for instance the process for handling purchase orders, is executed by branches of these organizations all over the world. And although the overall process is similar, local differences exist due to local regulations, cultural differences, etc. Moreover, most of these multinational organizations operate under multiple brands, targeting different markets. The business processes supporting these different brands are very similar, often with only minor differences. Hence, the information systems supporting the business processes have a lot in common, but are not exactly the same.

The information systems supporting these business processes of different brands or locations can be run locally, or centrally from the cloud. Because of the local differences in the execution of the processes, different configurations of the information systems are required. These differences are not only the language and currency used by the system, but the process models describing the process are also different [2, 3]. However, currently there is little to no support for sharing a common process model with variations.

Most information systems provide freedom during the execution of a process, for instance data-driven systems such as ERP and case management systems. At the same time these information systems keep detailed records of the execution of these processes. Process mining provides techniques to analyze the recorded behavior, and compare it with the modeled process. However, almost no process mining techniques exist to analyze the commonalities and differences between similar processes with minor variations.

1.1 Process Mining

By using historical facts, as recorded by the information system, process mining provides detailed insights into the process execution. An overview and positioning of process mining is shown in Figure 1.1. The general view is that process models aim to describe the 'real world'. These process models are used to configure the information system that should support this 'real world' process. While executing the defined process using the information system, historical records of the executed process are kept. This event data, in the form of event logs, is the main input of process mining analysis. Process mining provides links between the actual observed process execution and the modeled process behavior.

Three main classes of process mining techniques can be identified: (a) the *discovery* of new process models based only on the event log, (b) *conformance* verification of the recorded behavior with a provided process model and (c) *extension* of existing process models using the information from the event log.

Table 1.1 shows an excerpt of an example dataset used for process mining. Each row in the table represents one event and each column represents an attribute of this event. Events are associated with cases, and in Table 1.1 the events are already grouped by case and sorted chronologically. The first recorded event is related to case 1 and represents the execution of the activity `Register application` by Pete on December 30, 2010. Additional attributes can be related to this event such as the incurred cost, data attributes entered, etc. Events need to be uniquely identified, which is achieved by assigning unique identifiers. It is important that each event is related to a case and that events are sorted. In general the timestamp of execution is used to sort events chronologically. The times shown in Table 1.1 should be interpreted as the time when the corresponding activity was completed. In general events can also be recorded to register when activities are started, paused and resumed, etc.

Within the area of process mining the main focus of research has been on process discovery. The goal of process discovery is to, using only the behavior as recorded in the event log, construct a process model describing the underlying behavior. Although this aspect of process mining has received a lot of attention, and quite a few algorithms currently exist that do this, process discovery remains a challenge. Based on the event data of Table 1.1 for instance, a process discovery algorithm can discover that the process always starts by executing the activity `Register application`. Then the three activities `Check credit`, `Calculate capacity` and `Check system` are executed in different orders. Moreover, the activity `Check system` is not observed for cases 2 and 4. These activities are followed by either the activity `Accept request` or `Reject request`, and the process concludes with `Send decision e-mail`. Extracting these (control-flow) constructs from observed behavior only is not straightforward, especially since the event log might contain exceptions that should not be included in the process model and at the same time the observed behavior can be incomplete. Furthermore, the event log only contains examples of allowed behavior, there is no record of behavior that could not have occurred.

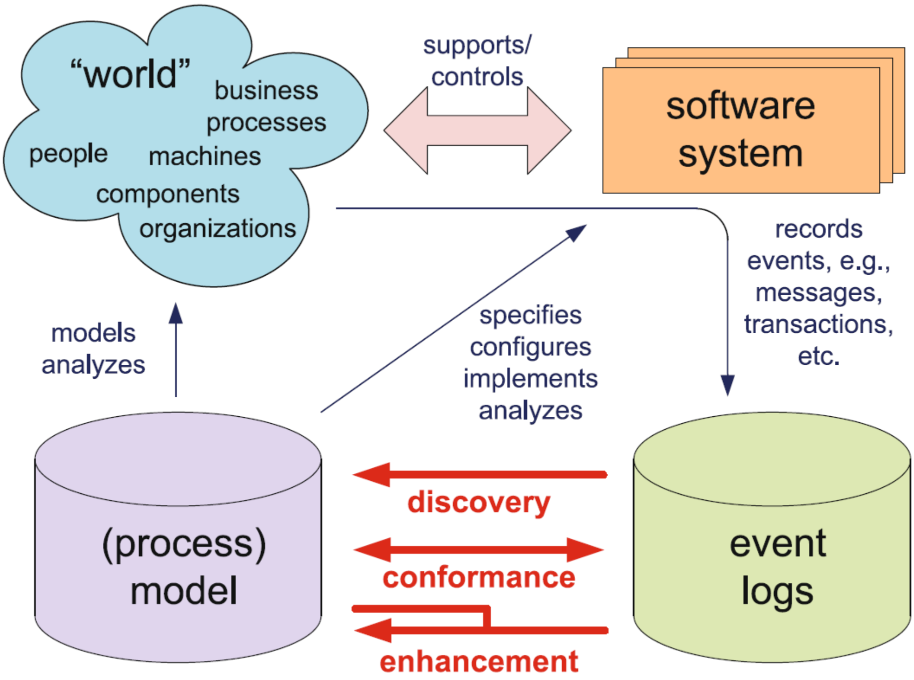


Figure 1.1: Positioning of Process Mining (from [5]).

Table 1.1: Example event data (adapted from [5]).

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010 11:02	Register application	Pete	50	...
	35654424	31-12-2010 10:06	Check credit	Sue	400	...
	35654425	05-01-2011 15:12	Calculate capacity	Mike	100	...
	35654426	06-01-2011 11:18	Check system	Sara	200	...
	35654427	07-01-2011 14:24	Reject request	Pete	200	...
	35654427	08-01-2011 09:03	Send decision e-mail	Pete	200	...
2	35654483	30-12-2010 11:32	Register application	Mike	50	...
	35654485	30-12-2010 12:12	Calculate capacity	Mike	100	...
	35654487	30-12-2010 14:16	Check credit	Pete	400	...
	35654488	05-01-2011 11:22	Accept request	Sara	200	...
	35654489	08-01-2011 12:05	Send decision e-mail	Ellen	200	...
3	35654521	30-12-2010 14:32	Register application	Pete	50	...
	35654522	30-12-2010 15:06	Check system	Mike	400	...
	35654524	30-12-2010 16:34	Check credit	Ellen	100	...
	35654525	06-01-2011 09:18	Calculate capacity	Sara	200	...
	35654526	06-01-2011 12:18	Accept request	Sara	200	...
	35654527	06-01-2011 13:06	Send decision e-mail	Sean	400	...
4	35654641	06-01-2011 15:02	Register application	Pete	50	...
	35654643	07-01-2011 12:06	Check credit	Mike	100	...
	35654644	08-01-2011 14:43	Calculate capacity	Sean	400	...
	35654645	09-01-2011 12:02	Reject request	Sara	200	...
	35654647	12-01-2011 15:44	Send decision e-mail	Ellen	200	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

This, combined with the fact that in general the original process model is not known, provides for an interesting and currently not sufficiently solved challenge.

The event data shown in Table 1.1 illustrates typical information present in an event log. However, depending on the applied techniques, more abstract views on the data are used, ignoring certain attributes for instance. In the remainder of this thesis, we are mainly concerned with the activity order, i.e., in which order are activities executed for cases. Although order is often determined based on the recorded timestamp, the timestamp itself can be discarded for most process discovery applications. Therefore, from here on, we often represent event logs by displaying activity sequences and their occurrence frequency.

To evaluate the quality of the discovered process models, four quality dimensions exists that relate the observed behavior with the process model. These four quality dimensions are shown in Figure 1.2. The quality dimension of *replay fitness* evaluates how well the observed behavior of the event log can be replayed by the process model. *Precision* evaluates how precise the process model describes the observed behavior. The less behavior the process model allows that is not observed in the event log, the more precise the process model describes the behavior. The third dimension of *generalization* evaluates whether the process model is not too specific for the observed behavior, but actually describes the process generating the observed behavior. Finally, the quality dimension of *simplicity* evaluates how simple, or human-readable, a process model is.

Many of the techniques developed in the context of process mining are available in the process mining framework ProM. As of version 6.4 ProM contains more than 120 packages and over 500 plug-ins. ProM can be obtained from `www.promtools.org`.

1.2 Introduction of Running Examples

Throughout this chapter, and thesis, we use a small collection of running examples to demonstrate several aspects and issues of process discovery. In this section we introduce these running examples in detail.

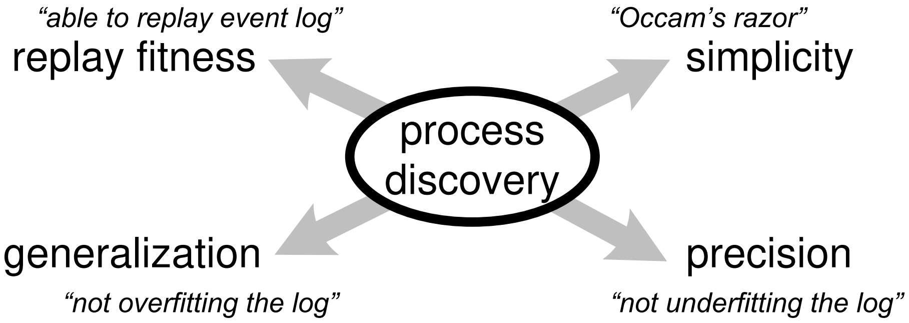


Figure 1.2: The four quality dimensions used to qualify a process model given the observed behavior (from [5]).

1.2.1 System with two Event Logs

The first running example consists of a system model as shown in Figure 1.3. From this model two event logs are simulated, shown in Table 1.2. The process model of Figure 1.3 describes a simple loan application process of a financial institute which provides small consumer credits through a webpage. When a potential customer fills in a form and submits the request from the website, the process starts by executing activity a which registers the application in the system and notifies the customer of the receipt of the request. Next, the following actions are performed in parallel. The credit is checked (activity b), the capacity is calculated (activity c) and the system is checked (activity d). However, the last activity of checking the system can be skipped in some cases. Next, a decision is made to either accept (activity e) or reject (activity f) the loan application. Finally the customer is informed of the decision by executing activity g.

From the system model two event logs are simulated. The event log of Table 1.2a contains only traces directly generated by the system model. This event log contains 11 unique traces spread over 100 traces in total. It is important to note that the process model allows for 20 unique traces in total, therefore the event log does not describe all possible behavior. Moreover, some traces occur more frequently in the event log than others, which emphasizes particular observed behavior. This is common in event logs since they often represent a tiny fraction of all the possible behaviors of the system, resulting in many of the recorded traces being unique.

Additionally an event log is generated that also contains exceptional behavior, i.e., behavior that is not possible according to the system model. This can be allowed behavior, non-allowed behavior or incorrectly recorded behavior. The

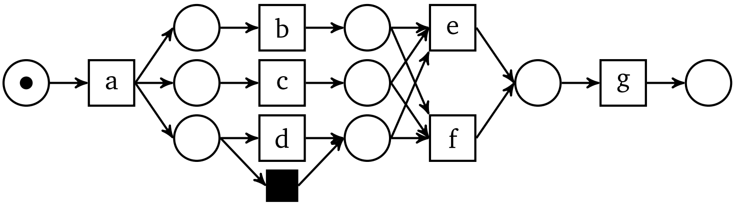


Figure 1.3: Process model as executed in the system, represented by a Petri net (a = register application, b = check credit, c = calculate capacity, d = check system, e = accept, f = reject, g = send decision e-mail).

Table 1.2: Running example used to discover a process tree.
(a = register application, b = check credit, c = calculate capacity, d = check system, e = accept, f = reject, g = send decision e-mail).

(a) Running example event log.

Trace	#
a b c d f g	38
a b d c f g	26
a b d c e g	12
a b c f g	8
a b c d e g	6
a d c b f g	4
a c d b f g	2
a c b e g	1
a d b c f g	1
a d b c e g	1
a c b f g	1

(b) Running example event log with exceptional behavior.

Trace	#	Trace	#
a b c d f g	380	a c d b f g	20
a b d c f g	260	a c b f g	10
a b d c e g	120	a c b d g	4
a b c f g	80	a d e g	4
a b c d e g	60	a b c g	3
a d c b f g	40	a c f g	3
a c b e g	10	a b c d e f g	2
a d b c f g	10	a b d e g	2
a d b c e g	10	a c d f g	2

resulting event log is shown in Table 1.2b. This event log consists of 1,020 traces. The first 1,000 traces are taken from the event log without exceptional behavior, which are replicated 10 times. The last 20 traces of the event log do not perfectly fit the process model of Figure 1.3. Hence, they describe infrequent, but exceptional behavior.

1.2.2 Four Similar Processes

The second running example consists of a collection of four process variants, based on the previous running example. For each of the four variants a process model is known, as shown in Figure 1.4, and corresponding event logs are generated and shown in Table 1.3. Variant 1 is exactly the previous running example, using the event log without exceptional behavior (cf. Table 1.2a). The other three variants are more simple variants of this process. Variant 2 for instance has a fixed execution sequence for checking the credit, which is split into sending (activity b1) and processing (activity b2) the request, calculating the capacity (activity c) and then checking the paper archive (activity d2). This variant allows for only two sequences, which differ in either accepting (activity

Table 1.3: Event logs of the four loan application process variants. (a = register application, b = check credit, b1 = send credit check request, b2 = process credit check reply, c = calculate capacity, d = check system, d2 = check paper archive, e = accept, f = reject, g = send decision e-mail).

(a) Event log for variant 1

Trace	#
a b c d f g	38
a b d c f g	26
a b d c e g	12
a b c f g	8
a b c d e g	6
a d c b f g	4
a c d b f g	2
a c b e g	1
a d b c f g	1
a d b c e g	1
a c b f g	1

(b) Event log for variant 2

Trace	#
a b1 b2 c d2 f	50
a b1 b2 c d2 e	20

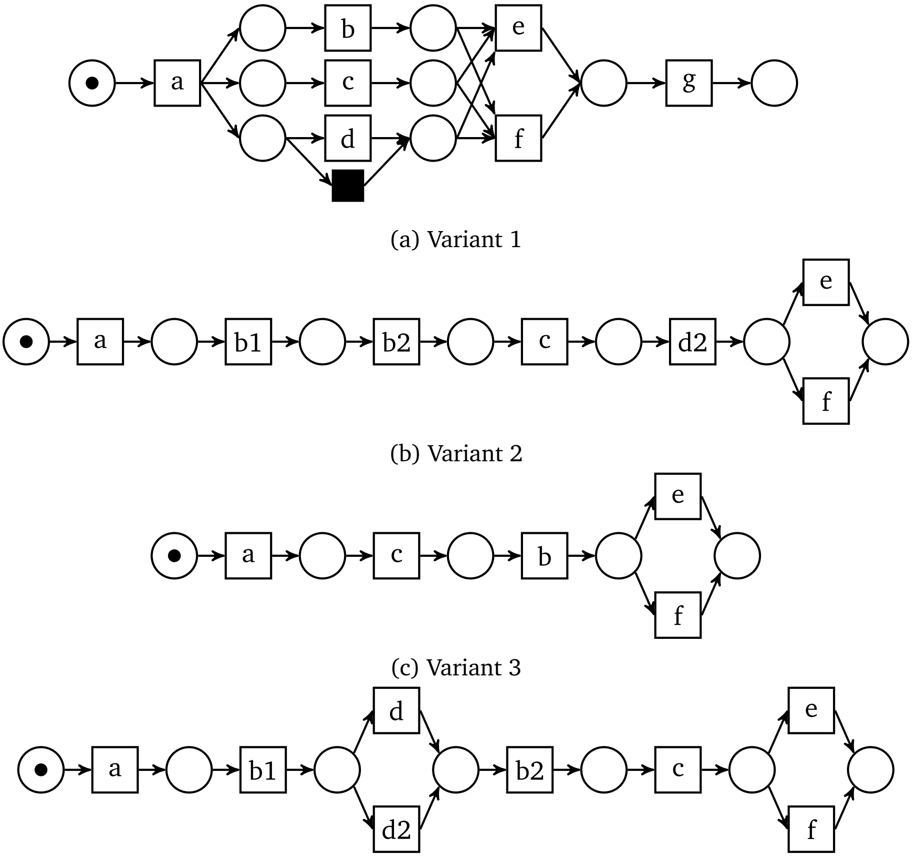
(c) Event log for variant 3

Trace	#
a c b e	120
a c b f	80

(d) Event log for variant 4

Trace	#
a b1 d2 b2 c f	60
a b1 d b2 c e	45

e) or rejecting (activity f) the application. Variant 3 also allows for just two different sequences and does not check the credit (activity d). Variant 4 also has the credit check split in sending (activity b1) and processing (activity b2) the request. It also includes both variants of checking the archive (digitally by executing activity d or manually through the paper archive by executing d2). This variant allows for four sequences in total, but only two are actually observed, because whenever the digital archive is searched, the application is accepted, while if the paper archive is checked the application is always rejected.



(a) Variant 1

(b) Variant 2

(c) Variant 3

(d) Variant 4

Figure 1.4: Petri net process models for the four loan application process variants (a = register application, b = check credit, b1 = send credit check request, b2 = process credit check reply, c = calculate capacity, d = check system, d2 = check paper archive, e = accept, f = reject, g = send decision e-mail). Note that activities b1 and b2 are more detailed versions of activity b. Activity d and d2 are two ways to achieve the same goal: check the archive.

1.3 Challenges in Process Mining

Process mining has gone through a fast development and growth over the past two decades. However, many challenges exist that need to be addressed. The process mining manifesto [11], published by the IEEE Task Force on Process Mining, lists several challenges and guiding principles for process mining. In this section we discuss a selection of these challenges and guiding principles in detail, and propose new challenges that should be addressed. These challenges can be divided into two categories. The first category of challenges is related to process discovery. The second category addresses the lack of techniques that support the comparison of process executions. Before we address concrete challenges, we first discuss the process discovery results of existing algorithms on the running examples introduced in Section 1.2.1.

1.3.1 Results of Existing Process Discovery Techniques

Table 1.4 shows a qualification of the results of existing process discovery algorithms on the running example event logs of Table 1.2. The results are evaluated using five criteria. The first criterion is whether the discovered process models are error-free, i.e., can be executed without errors. The results show that half of the algorithms actually produce error-free models on the running example data. The four other algorithms created process models that are 'relaxed' error-free (they can finish but work remains), indicated by the yellow square. However, none of these algorithms actually guarantee to always produce (relaxed) error-

Table 1.4: Comparison of results of process discovery algorithms on the running example event logs (f=replay fitness, p=precision, g=generalization, s=simplicity).

Algorithm	Error-free?	f	p	g	s
α-algorithm	✓	□	✓	✗	✗
Genetic miner	□	□	□	✗	✗
Heuristics miner	□	✗	✓	✗	✗
ILP miner	□	✓	✗	✓	✗
Inductive miner	✓	✓	✗	✓	✓
Language-based region theory	✓	✗	✗	✗	✗
Multi-phase miner	□	✓	✗	✗	✗
State-based region theory	✓	✓	✗	✗	✗

free process models.

The resulting process models are also evaluated using the four quality dimensions: replay fitness ('f' column), precision, ('p' column), generalization ('g' column) and simplicity ('s' column). The main observation is that most process discovery algorithms perform well on only one quality dimension.

This comparison is based on experimental results discussed in more detail in Section 6.9. In the remainder of this section we discuss this table in more detail.

1.3.2 Challenge 1: Produce Correct Process Models

A (discovered) process model is often used for more than just documentation. Many analysis techniques can be applied on a process model, such as process validation, process optimization and simulation. However, most of these techniques only work if the process model is correct [69,133], i.e., free of structural errors such as deadlocks and live locks or improper termination. The correctness of a process model is therefore crucial for the usability of the process model. However, very few of the existing process discovery algorithms guarantee to produce a correct process model, as is illustrated in Table 1.4. This means that in many cases, and especially for real life event logs, the results produced by these algorithms cannot be used for further analysis.

Using the simple event log of Table 1.2a, this challenge can be demonstrated using the heuristics miner. When the heuristics miner is applied to this event log, the process model as shown in Figure 1.5 is the result. This process model is incorrect since after executing the sequence $\langle a, c, e, g\rangle$, the process model should be completed, because there is a token in the last place. However, there is still a token remaining before or after activity `b`, and there might be a token remaining before or after activity `d`. Therefore, work is still pending even though the process model is completed. Even on a very simple example event log, some process discovery algorithms, not only the heuristics miner, fail to produce correct process models. On many real life event logs even more process discovery algorithms fail.

1.3.3 Challenge 2: Separation of Visualization and Representational Bias

Challenge 5 of the process mining manifesto [11] states that "*it is important to separate the visualization of the result from the representation used during the* *actual discovery process*". Furthermore, it states that the representation used internally by a process discovery algorithm should be a conscious choice, and not only be driven by the preferred graphical notation.

Most process discovery techniques discover Petri nets, such as the result shown on the left in Figure 1.6. Although used frequently in academia because of their formal semantics, Petri nets are not the preferred modeling notation in industry. And even though Petri nets are very expressive, their graphical representation is not always compact. Other process discovery algorithms use special classes of Petri nets to represent their results, such as the result shown on the right in Figure 1.6. However, most of these classes, such as elementary nets, are very hard to interpret. Although most algorithms consciously choose a particular notation that best suits their approach, this is often not the best way to visualize the result.

There are also process discovery algorithms that use their own representational bias to communicate their result. Examples of such process modeling notations are causal nets [9], heuristics nets [180,181] and fuzzy models [96,97]. Although these choices are conscious ones, better ways to visualize and communicate the discovered process model often exist.

The problem with all these notations is that they do not represent the results of the algorithm to the end-user in a suitable way. The process models shown in Figure 1.6 for instance describe the observed behavior quite well. However, the resulting process models are very hard to interpret, even for an expert user. Most end-users are not familiar with these notations and would rather see a process model in BPMN [143] or EPC [159] notation, as used in industry. Therefore, there should be a separation between the representational bias used by the algorithm to construct a process model, and the visualization of the resulting process model.

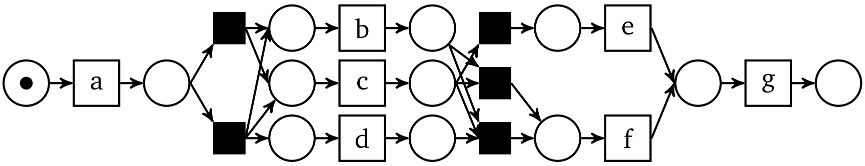


Figure 1.5: Result of the heuristics miner on the running example event log of Table 1.2a. The process model is incorrect since work ('tokens') can remain even though the process model is completed.

1.3.4 Challenge 3: Balance the Quality of Discovered Process Models

As mentioned in Challenge 6 in the process mining manifesto [11], event logs are far from complete, and may contain exceptional behavior (‘noise’). The four quality dimensions of Figure 1.2 are used to quantify the quality of the process model using the given event log. All process discovery algorithms make assumptions regarding the event log, and the emphasis of the different quality dimensions, as is shown in Table 1.4. Some algorithms, such as the Fuzzy miner [96], purposely simplify the process model, even if this means that not all behavior is described. Other algorithms always produce process models that describe all observed behavior, such as the ILP miner, resulting in process models that are very complex and/or meaningless. Additionally, most algorithms assume that there is exactly one (perfect) process model that describes the event log.

However, given an event log, there is no single perfect process model that can be discovered. What is considered to be the ‘best’ process model for a given event log depends on many factors, including intended use of the model and characteristics of the event log. Some algorithms allow for some parametrization, resulting in different process models, but these algorithms do not communicate clearly what the effect of the parameters is on the quality of the resulting process model. Typically, algorithms produce a single process model as a result, instead of providing insights into the trade-offs between the different quality dimensions.

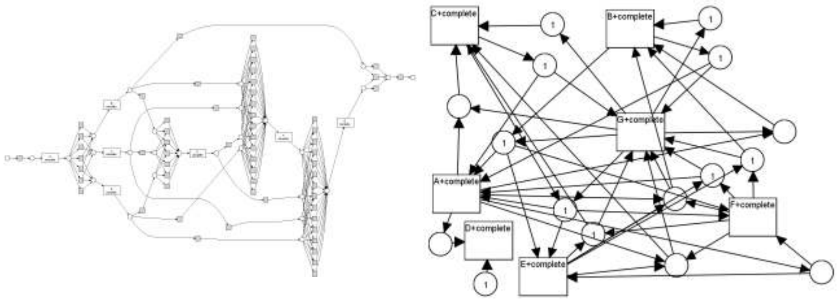


Figure 1.6: Two example process models, discovered from the running examples without and with exceptional behavior respectively, demonstrating an unsuitable visualization to the end user.

Consider for instance the process model of Figure 1.3 and the process model of Figure 1.7 for the event log of Table 1.2a. Both process models describe a process that starts with activity a and ends with activity g. However, the description of the other activities differs significantly. The process model of Figure 1.3 is able to replay all of the behavior of the running example event log of Table 1.2a, but also allows for additional behavior. On the other hand, the process model of Figure 1.7 describes the event log in a precise manner, i.e., does not allow for additional behavior, but at the same time cannot explain all of the observed behavior. This simple example shows that for a given event log several process models can be discovered, and that in general none can be classified as 'best'.

1.3.5 Challenge 4: Improve Understandability for Non-Experts

The purpose of process mining is to produce process models that can be used for further analysis, and not merely for documentation. Therefore, Challenge 11 of the process mining manifesto [11] states that the understandability of the results for non-experts should be improved. The results of an algorithm are only useful if the user, who most likely is a non-expert in the field of process mining, is able to use the results. One way to ensure understandability is to use a suitable representation (see Challenge 2). However, it is also important to illustrate the trustworthiness of the result. Almost all algorithms always produce a process model, even if there might be very little data to support this description of behavior. The quality of the resulting process model should always clearly be indicated, preferably by using the four quality dimensions. This helps users, both experts and non-experts, draw correct conclusions from discovered process models.

Consider again the process models of Figure 1.3 and Figure 1.7 as a description of the observed behavior of Table 1.2a. These two process models alone

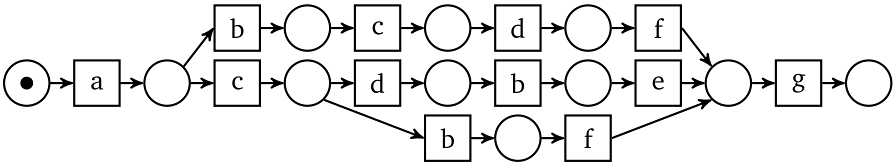


Figure 1.7: Alternative process model for the running example of Section 1.2.1.

are not enough for a user to decide how trustworthy these process models are. Additional information is required, for instance the scores for each of the four quality dimensions. The process model of Figure 1.3 has a perfect replay fitness and simplicity, a precision of 0.897 and a generalization of 0.870. The process model of Figure 1.7 has a perfect precision and simplicity, a replay fitness of 0.885 and a generalization of 0.671. Using this information the user can understand the differences between the process models better. Furthermore, the quality scores show that there is enough data to support both discovered process models and that each provides different quality trade-offs.

1.3.6 Challenge 5: Use Existing Knowledge in Process Discovery

A challenge not explicitly mentioned in the process mining manifesto [11] is that existing knowledge should be reused by process mining algorithms, and process discovery algorithms in particular. The behavior observed in an event log usually originates from a system that is configured to support a particular business process. The configuration phase is usually supported by documentation, e.g. business process models. Although discovering a process model by using only the observed behavior can be useful, indicating the observed deviations from the documented process model provides additional analytical insights. Moreover, instead of only showing deviations, algorithms should be made available to repair a given process model using observed behavior. By allowing for various gradations of repair, the process owner can investigate how far the modeled behavior is from the actual observed behavior. This makes it clear what the exact differences are between the modeled and observed behavior.

Consider for instance the process model as shown in Figure 1.7 for the event log shown in Table 1.2a. If we assume that this is the process as it is known and documented in the organization, then we can compare the observed behavior with this process model. Many interesting insights can be obtained, for instance that the observed behavior does not always fit the documented process, which indicates that the system allows for more behavior. Additionally, the documented process model assumes that activity e (acceptance of the application), can only occur after activity b (check credit). The observed behavior does not always contain this relation, hence some internal rules might be violated. In the end the organization has two options: either modify the system to disallow the undesired behavior, or update the (documented) process model to better reflect reality.

1.3.7 Challenge 6: Describe a Family of Processes

Almost all process discovery techniques, and process mining techniques in general, only consider a single event log and/or process model in isolation. Challenge 7 of the process mining manifesto [11] states that more attention needs to be given to cross-organizational process mining. As mentioned in the introduction of this chapter, and as will be further detailed in the discussion of the CoSeLoG project in Section 1.4, many organizations execute very similar processes. The event logs from these processes can be seen as describing a family of processes. Very few techniques currently exist that are capable of taking multiple event logs as input, and produce a single process model that describes a family of processes as output. However, there are many use cases for this scenario, where this type of process model provides valuable information.

A description of the family of processes does not only add value in the setting where different organizations execute similar processes. One could also split the observed behavior based on the customer type (e.g. ‘silver’ versus ‘gold’ member), and then compare the differences in process execution. Numerous examples can be thought of, for instance: splitting by time period (e.g. year or season), by the employee responsible for the case, or by the communication channel the customer used for his application (e.g. physical, telephone or e-mail). All these very similar, yet different processes, can be described by a single model, that also indicates where the various process variants differ.

When considering the four process variants as discussed in Section 1.2.2, one can see the similarities and differences clearly. For instance, one can observe that the behavior of variant 3 is also described by the process model of variant 1. It should be possible to create a single process model that, using configurations, describes both processes. An example of such a process model is shown in Figure 1.8. The process model can be configured to describe variant 1 by removing the arrow connecting activity c with the place after activity b, as well as removing the silent action next to activity g. This enables activity c to be executed in parallel to activities b and d, and activity g cannot be bypassed. The configuration for variant 3 consists of removing activities d and g.

In a similar way a configurable process model for variants 2 and 4 could be created. Moreover, one process model might exist that is able to describe all four variants. Process mining techniques can be extended to discover such a process model that describes a family of event logs.

1.3.8 Challenge 7: Compare Similar Observed Behavior

Many process mining techniques exist that provide different insights into the observed behavior, without discovering a process model (e.g. the dotted chart [165], social network analysis [166] and trace alignment [41]). However, few techniques exist that allow for the comparison of similar observed behavior. By visualizing the observed behavior without direct use of a process model, different insights can be gained into the (dis)similarities between process executions.

Consider for instance the comparison table shown in Table 1.5. The table compares the four event logs with four configurations of a configurable process model (as discussed in the previous section). Currently no technique exists that is able to compare information considering multiple event logs, and possibly process models. Without going into detail, the green cell in the middle of Table 1.5 indicates the replay fitness score of the event log on the process model (configuration). This shows for instance that event log 3 can be replayed very well on the configuration for organization 1. This type of analysis technique facilitates organizations that want to seek closer collaboration and want to base their analysis on more than just documented process models.

1.3.9 An Algorithm that Addresses all Challenges

The first four challenges presented in this section discuss fundamental challenges for process discovery. Challenges 5 and 6 present extensions to process

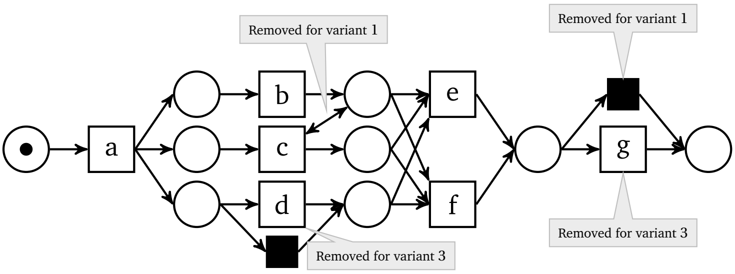


Figure 1.8: Configurable process model for variants 1 and 3 of Figure 1.4. In order to obtain variant 1 the edge between activity c and the place after activity b, as well as the option to skip activity g are removed. To obtain the behavior of variant 3 activities d and g are removed. The edges connecting the removed activities are also removed.

Table 1.5: Example of a way to compare the four process variants (see Table 10.3). The higher the central value, the better the process model variant describes the behavior of the event log.

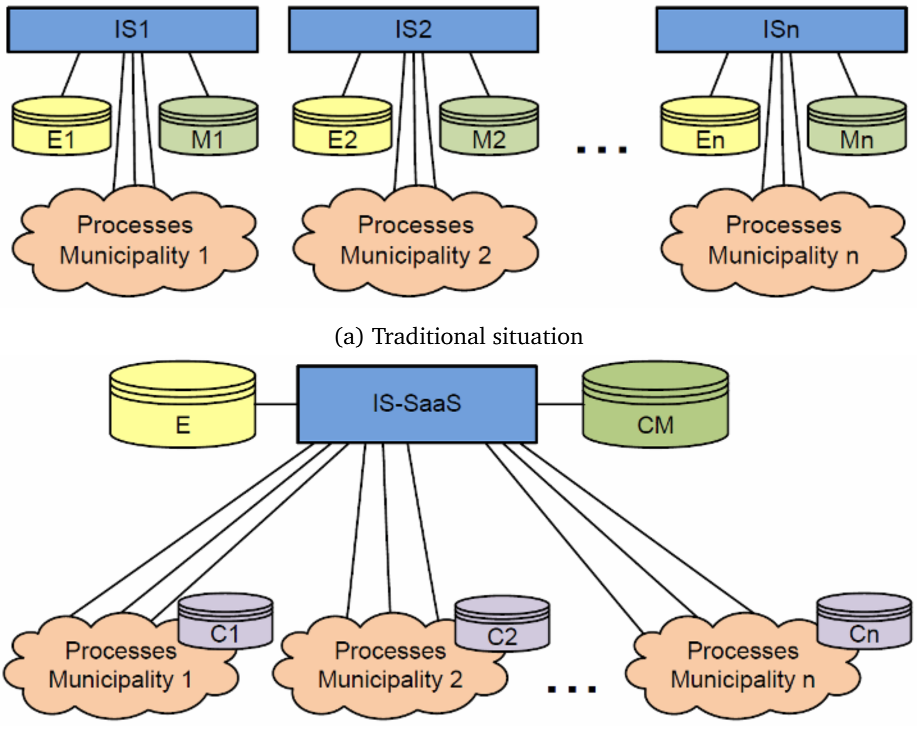
	Config 1	**Config 2**	**Config 3**	**Config 4**	**Log Stat**
Event Log 1	1.000	0.506	0.575	0.580	100
Event Log 2	0.525	1.000	0.553	0.833	70
Event Log 3	0.933	0.656	1.000	0.656	200
Event Log 4	0.579	0.833	0.553	1.000	105
Model Stat	12	9	7	11	

discovery and Challenge 7 proposes an extension to process mining analysis techniques. When addressing the latter three challenges the first four fundamental challenges should also be addressed. In order to do so, a proposed solution should be able to incorporate the fundamental challenges while being able to be extended in order to address the other three challenges. Additionally, possible future scenarios should also be supported. This affects the choice for the type of algorithm used and the internally used process model notation. It is also important to observe that especially Challenge 3, balancing the quality of the discovered process models, requires flexibility. In this thesis we therefore present a flexible framework using an evolutionary algorithm approach (see Chapter 4) to develop algorithms that address the challenges.

1.4 The CoSeLoG Project

A clear example of organizations that have similar processes are municipalities. As of January 1, 2014 there are 403 municipalities in the Netherlands [95]. It is estimated that each municipality offers between 400 and 500 different products and services, such as driver licences, building permits, subsidies, citizen administration, and health care support. National rules and regulations apply for most of these products and services. Still, each municipality defines its own processes to support the production and delivery of their products and services. Although each municipality is different in size, organizational culture, etc., there are still many commonalities in the way processes are executed. Municipalities have the additional advantage that they can seek collaboration with other municipalities since they are not direct competitors. This allows municipalities to share knowledge and infrastructure and thus reduce costs. However, at the same time they want to retain their identity and visibility to their inhabitants. This implies that each municipality has different requirements and priorities for their business processes. One municipality for instance would like to work as cost-efficiently as possible, while another municipality considers quality of their service as more important. This influences the business process, for instance by the number of quality checks implemented.

The *Configurable Services for Local Governments (CoSeLoG)* research project



(a) Traditional situation

(b) Envisioned situation

Figure 1.9: Traditional situation and envisioned situation within the CoSeLoG project (IS=Information System, M=Process Model, E=Event Log, IS-SaaS=Information System as Software-as-a-Service, CM=Configurable Process Model, Cn=Configuration) (from [2]).

[2,47,59] aims at harmonizing processes between municipalities, while at the same time providing freedom of choice to each municipality. Within this project the commonalities between processes of municipalities are used to develop a shared business process management system in a shared software-as-a-service environment [2,3]. Figure 1.9a depicts the traditional situation with municipalities. Each municipality purchases, configures and runs an information system to support one or more of their business processes. This information system is configured with the (intended) business process models, which are used to execute the processes within the organization. Here, each municipality mainly works in isolation. Each municipality translated the requirements of the law for a process to a corresponding process model, implemented this model and started executing it. Within the CoSeLoG project we envision a situation where municipalities, and organizations in general, can share an information system, as is shown in Figure 1.9b. This information system (IS-SaaS) is provided using the Software-as-a-Service paradigm, i.e., the software is hosted centrally and municipalities have to have a subscription to use it. However, even though the processes of the individual organizations have a lot in common, local deviations still exist and should be supported. Therefore, the shared information system uses a process model that can be individualized using configurations of the shared process model. This reduces the effort, and hence cost, of maintaining the process model, especially when regulations change [2,3]. Moreover, the solution is cheaper since infrastructure and management costs are shared.

The original information system keeps a record of historical process data, recorded in event logs. This data can be used to analyze the current execution of processes by use of process mining. This is necessary for the transfer of municipalities to the envisioned situation, to ensure continuing support for their processes.

Once organizations use the IS-SaaS system, the commonly shared event log allows for easy comparison of behavior between organizations. We envision that organizations will use this information to learn from each other and share knowledge. This will lead to a natural way of standardizing the process models, where the municipalities voluntarily move towards a small number of configurations of the configurable process model.

Within the CoSeLoG project we collaborate with different partners from industry. Ten participating Dutch municipalities (Bergeijk, Bladel, Coevorden, Eersel, Emmen, Gemert-Bakel, Hellendoorn, Oirschot, Reusel de Mierden, and Zwolle) provide use-cases, concrete process models and event data, and requirements from a user perspective. A Dutch cooperation of municipalities for IT services (DiMPACT) and a commercial IT partner (Perceptive Software) provide insights from a software provider's point of view.

Within the CoSeLoG project five municipality processes have been investigated in detail:

1. Processing applications for a receipt from the municipality's basic administration ('uittreksel Gemeentelijke Basis Administratie (GBA)' in Dutch);

2. Dealing with reports regarding the public area ('Melding Openbare Ruimte (MOR)' in Dutch);

3. Processing applications for building and/or environmental permits ('Wet Algemene Bepalingen omgevingsrecht (WABO)' in Dutch);

4. Processing applications for social services ('Wet Maatschappelijke Ondersteuning (WMO)' in Dutch);

5. Handling objections against house taxation ('Wet Waardering Onroerende Zaken (WOZ) bezwaar' in Dutch).

These processes differ in complexity and number of resources, parties and activities involved, and the number of cases handled per year. For each of the ten participating municipalities, each of these five processes were investigated. If available, the execution history was obtained from the information system used to support the process. This allowed for the analysis of these processes using process mining techniques.

The challenges as discussed in Section 1.3 all played a role in the CoSeLoG project. Since processes cannot easily be merged into one another, analyzing the observed behaviors helps in finding commonalities and differences between municipalities. However, first basic process discovery techniques can be applied to provide insights into the current (traditional) situation, before organizations move towards the shared infrastructure. Here Challenges 1 through 5 play a role, as they do in any process discovery project. Solutions to challenge 6 can be applied to discover the configurable process model that is used to configure the IS-SaaS, using the individual event logs from the current information systems. Using the solutions for Challenge 7 insights into the processes can be provided to participants, based on the similar behavior as recorded in either the traditional or envisioned situation. Solutions for all challenges are crucial for the adoption of the IS-SaaS system, since the process model only describes the documented process. Therefore, process execution data needs to be considered to ensure that organizations can successfully move to the IS-SaaS system.

1.5 Contributions and Structure of this Thesis

The contributions of this thesis can be summarized as follows:

1. *A guaranteed error-free process model notation.* (Chapter 3, addresses Challenges 1 and 2)
   In order to be useful, any process model should be internally consistent, i.e., have no deadlocks, have no livelocks and be able to terminate properly. An evaluation of existing process modeling notations however reveals that very few notations can guarantee this. Therefore we present a process modeling notation that is guaranteed to be free of such errors. We show that our process modeling notation can be easily translated to error-free process models in other notations for further use.

2. *A detailed discussion of the four quality dimensions in process discovery.* (Chapter 5, addresses Challenges 3 and 4)
   The importance and interrelationship of the four quality dimensions used in process discovery have never been thoroughly investigated. In this thesis we show the influence of each of these quality dimensions on the discovered process model. We also argue that all four quality dimensions need to be taken into account. At the same time we argue that the dimension of replay fitness is the most important one, since it relates the observed behavior to the process model.

3. *A flexible and extensible process discovery algorithm.* (Chapter 4 and Chapter 6, addresses Challenges 3, 5 and 6)
   The main contributions of this thesis are a process discovery framework and process discovery algorithm that are both flexible and extensible. Flexibility is provided by the ability of the algorithm to emphasize certain process model qualities, as preferred by the user. The four quality dimensions used for evaluation of discovered process models are incorporated. However, additional quality dimensions can easily be considered:

   (a) *Mediation between a given process model and observed reality.* (Chapter 8, addresses Challenge 5)
   The search for the 'best' process model is started from a given normative process model. An additional quality measure is added that evaluates the similarity between a discovered process model and the normative process model. This allows for process model improvement by changing a given process model, using the observed behavior.

   (b) *Discovery of a configurable process model.* (Chapter 9, addresses Challenge 6)
   A configurable process model is a process model that can be configured before run-time to (dis)allow certain execution paths. A configurable process model thus describes a family of process models. The process model notation presented in this thesis is extended to describe a configurable process model. An additional quality dimension is added that evaluates how good a configuration is, to find the best process model and corresponding configuration. This enables the process discovery algorithm to discover a configurable process model, describing a family of event logs.

4. *Inter-Organizational process comparison.* (Chapter 10, addresses Challenge 7)
   This thesis also describes a comparison framework to compare processes between organizations, by using both a collection of event logs and a collection of (corresponding) process models, possibly from a configurable process model. By visualizing behavior, differences and commonalities in the observed behavior of the processes can be detected without necessarily visualizing the process model itself.

The first part of this thesis provides an *introduction* to process mining and open challenges in its domain. Furthermore, preliminaries are provided in Chapter 2.

The second part discusses two fundamental concepts used in the remainder of this thesis. First *process trees* are introduced in Chapter 3 as a process modeling notation that is guaranteed to be error-free. Second, our *framework for flexible process discovery*, that applies of an evolutionary algorithm, is discussed in Chapter 4.

Part three discusses the different aspects of the flexible process discovery framework and an evaluation of the framework based on case studies. In Chapter 5 the *quality dimensions* commonly used in process discovery are discussed in detail. A *process discovery algorithm* implemented in the framework is discussed in Chapter 6. This process discovery algorithm is *evaluated using both artificial and real-life data* in Chapter 7.

The fourth part discusses extensions of the process discovery framework. Chapter 8 discusses how *observed and modeled behavior can be balanced* by the algorithm. The algorithm is extended to consider the behavioral records of *multiple organizations* in Chapter 9. A different view on *comparing behavior records* of different organizations is discussed in Chapter 10.

Part five concludes this thesis. In Chapter 11 the use and *implementation of the framework and algorithms* as implemented in the process mining toolkit ProM is detailed. Chapter 12 summarizes the main results and discusses possible research directions that build on the work presented in this thesis.

Chapter 2

Preliminaries

In Section 2.1 we introduce basic mathematical notations used throughout the remainder of this thesis. We then discuss the process modeling notations used in this thesis in Section 2.2. Section 2.3 introduces event logs.

2.1 Notations

In this section we introduce basic notations for sets, functions, sequences and bags as used in the remainder of this thesis.

We define sets as follows:

Definition 2.1 *(Sets)*
A set *is a possibly infinite collection of* elements*. We denote a finite set by listing its elements between braces, e.g., a set A with elements a, b and c is denoted as $\{a,b,c\}$. The* empty set*, i.e., the set with no elements, is denoted by $\emptyset$. To denote a non-empty set we write A^+. Let $A = \{a_1,\ldots,a_n\}$ be a* set *of size $n \in I\!N$, then $|A| = n$ denotes the size of set A.*

In the remainder of this thesis, we typically use uppercase letters to denote sets and lowercase letters to denote the elements of that set.

We define the union, intersection and difference operations on sets as follows:

Definition 2.2 *(Union, Intersection and Difference)*
Let $A = \{a,b,c,d\}$ and $B = \{a,c,d,e\}$ be non-empty sets. The union *of A and B,*

denoted $A \cup B$, is the set containing all elements that are in either A or B, e.g., $A \cup B = \{a,b,c,d,e\}$. The intersection *of A and B, denoted $A \cap B$, is the set containing elements that are both in A and B, e.g., $A \cap B = \{a,c,d\}$. The* difference *between A and B, denoted $A \setminus B$, is the set containing all elements of A that do not exist in B, e.g., $A \setminus B = \{b\}$.*

We define functions as follows:

Definition 2.3 *(Functions)*
Let A and B be non-empty sets. A function *f from A to B, denoted $f : A \rightarrow B$, is a relation from A to B, where every element of A is associated with an element of B. For all functions f, $Dom(f)$ and $Rng(f)$ denote the domain and range of function f respectively.*

We define the sequence, concatenation of sequences, and projection on sequences as follows:

Definition 2.4 *(Sequence)*
Let A be a set. A sequence $\sigma = \langle \sigma_1, \ldots, \sigma_n \rangle$ can be represented by listing its elements between angled brackets, where σ_i refers to the i-th element of the sequence and $|\sigma| = n$ denotes its length. $\langle\rangle$ denotes the empty sequence. A^* denotes all possible sequences from elements of the set A.

Definition 2.5 *(Concatenation)*
Concatenation of two sequences σ and σ' is denoted with $\sigma \cdot \sigma'$. Similarly, concatenation of an element $a \in A$ and a sequence σ is denoted $a \cdot \sigma$.

Definition 2.6 *(Projection)*
For all $A' \subseteq A, \sigma_{\downarrow A'}$ denotes the projection of a sequence $\sigma \in A^*$ on A', e.g., $\langle a,a,b,c, \rangle_{\downarrow \{a,c\}} = \langle a,a,c \rangle$.

We define multi-sets, also known as bags, as follows:

Definition 2.7 *(Multi-sets(bags))*
Let A be a set. A multi-set *M over A is a function $M : A \rightarrow I\!N$. $\mathbb{B}(A)$ denotes the set of all multi-sets over a finite domain A. We write e.g., $M = [a, b^2]$ for a multi-set M over A where $a,b \in A, M(a) = 1, M(b) = 2$, and $M(c) = 0$ for all $c \in A \setminus \{a,b\}$. The size of a multiset, denoted by $|M|$, is defined as $|M| = \sum_{a \in A} M(a)$.*

2.2 Process Models

Process models capture the behavior of a process and are thus an abstraction of reality, emphasizing certain aspects of a process. The importance of modeling business processes is illustrated by the plethora of process modeling notations, sometimes referred to as the new "tower of Babel" [5]. In this section we discuss two of the most commonly used low-level process modeling notations: *labeled transition systems* and *Petri nets*. Additionally, we discuss the high-level process modeling notation BPMN, which is commonly used in industry. In Section 3.2 we discuss several other proces modeling notations.

Process models describe if and in which order activities are to be executed. An activity is a well-defined step in the process. We use the notion of an *activity universe* to describe all possible activities, which we define as follows:

Definition 2.8 *(Activity Universe)*
Let $\mathscr{A}$ denote the activity universe, i.e., the universe of all possible activity names. Let $\mathscr{A}^\tau = \mathscr{A} \cup \{\tau\}$, where $\tau \notin \mathscr{A}$. The symbol τ represents a silent, or unobservable, action.

Process models are usually represented in terms of graphs and have a corresponding graphical representation. A graph consists of nodes and arcs that connect them. A directed graph is a graph whose edges have directions. In this thesis, we consider graphs whose arcs have both directions and labels. Such graphs are called *labeled directed graphs*. We formalize labeled directed graphs as follows:

Definition 2.9 *(Labeled Directed Graph)*
A labeled directed graph *is a tuple $DG = (NG, EG, LG)$ where NG is a set of nodes, LG is a set of labels, and $EG \subseteq NG \times LG \times NG$ is a set of labeled edges.*

2.2.1 Labeled Transition Systems

The most basic process modeling notation is a *transition system* [33]. A transition system consists of *states* and *transitions*. Figure 2.1 shows a transition system consisting of 13 states and 21 transitions. The states are represented by black circles. There is one initial state, marked by $s1$, and one final state, marked by $s13$. Transitions are represented by arcs between two states, and each transition is labeled with the name of an activity. Multiple arcs can have the same label.

We define a labeled transition system in the same way as in [5]:

Definition 2.10 *(Labeled Transition System)*
A labeled transition system *is a tuple* $TS = (S, A, T)$ *where* S *is the set of* states, $A \subseteq \mathscr{A}^{\tau}$ *is the set of* activities, *and* $T \subseteq S \times A \times S$ *is the set of* transitions. $S^{start} \subseteq S$ *is the set of* initial states, *and* $S^{end} \subseteq S$ *is the set of* final states.

The sets S^{start} and S^{end} are defined explicitly. In principle S can be infinite, however for most practical applications the state space is finite. In this case, the transition system is also referred to as a Finite-State Machine (FSM) or a finite-state automaton.

Transition systems are very expressive. Many process models with executable semantics can be mapped onto a transition system [5]. Thus, analysis techniques and notions defined for transition systems can be easily related to other languages such as BPMN, BPEL, EPC, and Petri nets.

However, transition systems cannot express parallelism in a concise way, since all possible sequences need to be explicitly modeled. For example, if n activities are in parallel, this results in $n!$ possible execution sequences. The corresponding transition system requires 2^n states and $n \times 2^n - 1$ transitions. Therefore more powerful process modeling notations are required, such as Petri nets which can express concurrency in a more efficient way.

2.2.2 Petri Nets

One of the best investigated process modeling languages that supports concurrency are Petri nets [139]. Petri nets use a very simple notation of circles representing *places* and squares representing *transitions* with arrows connecting them in a bipartite manner. Transitions can represent a task and when executed they consume one *token*, presented by black dots, from each of their input places and

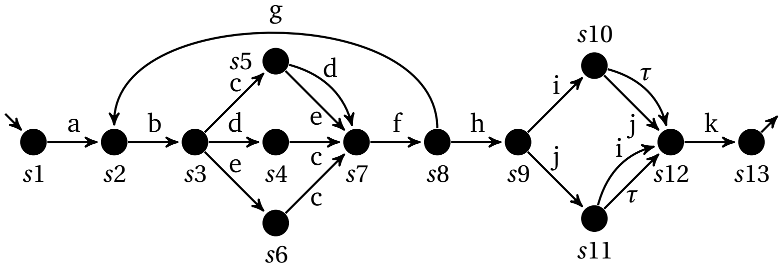


Figure 2.1: A labeled transition system with 13 states and 21 transitions.

produce a token in each of their output places. In this way, tokens are moved between places, and the distribution of tokens over the places indicates different states of the process model, and is called a *marking*. Special markings are the initial marking, which indicates how the process starts, and the final marking which indicates when the Petri net is in a terminate.

An example of a Petri net model is shown in Figure 2.2. The initial marking is $[p_0]$, which means that the model starts with a token in place p_0. By firing transition a the token is consumed from place p_0 and produced in place p_1. After transition b has fired, there are tokens in places p_2 and p_3, i.e., the marking is $[p_1, p_3]$, which enables two parallel branches. Of the transitions d and e only one can fire: they are in a so-called exclusive choice relation. After transition c has also fired, transition f is enabled and after firing consumes the tokens from places p_4 and p_5 and produces a token in p_6. Now again there is a choice, and if transition g fires, the token goes back to p_1. If transition h fires, both transition i and j are enabled. After transition i, transition j, or both, have fired, the token can continue to place p_{11}. This is done by firing a silent, or τ-transition, denoted by the filled black transitions. These transitions do not correspond to performing any activity and only distribute the tokens. The final marking $[p_{12}]$ is reached by firing transition k which places a token in p_{12}.

We define Petri nets in a similar way as in [7]:

Definition 2.11 *(Petri net, Marking)*
A Petri net *is a triplet* $N = (P, T, F)$ *where* P *is a finite set of* places*,* T *is a finite set of* transitions *such that* $P \cap T = \emptyset$*, and* $F \subseteq (P \times T) \cup (T \times P)$ *is a set of directed arcs, called the* flow relation*. A* marked *Petri net is a pair* (N, M)*, where* $N = (P, T, F)$ *is a Petri net and where* $M \in \mathbb{B}(P)$ *is a* multi-set *over* P *denoting the* marking *of the*

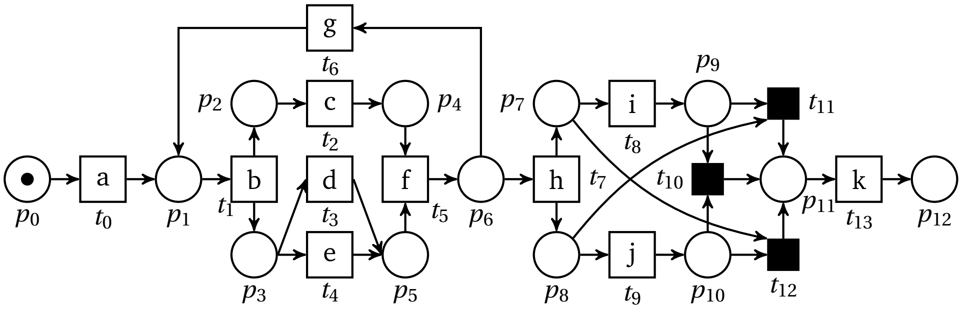


Figure 2.2: Example of a Petri net with initial marking p_0 and final marking p_{12}.

net.

A Petri net $N = (P, T, F)$ defines a directed graph with nodes $P \cup T$ and edges F. For any $x \in P \cup T$, $\bullet x = \{y | (y, x) \in F\}$ denotes the set of input nodes and $x\bullet = \{y | (x, y) \in F\}$ denotes the set of output nodes. A transition $t \in T$ is *enabled* in a marking M of net N, denoted as $(N, M)[t\rangle$, if each of its input places $\bullet t$ contains at least one token. For instance, in the Petri net of Figure 2.2, transition t_0 is enabled because all of its input places, in this case p_0, are marked.

An enabled transition t may *fire*, i.e., one token is removed from each of the input places $\bullet t$ and one token is produced for each of the output places $t\bullet$. $(N, M)[t\rangle(N, M')$ denotes that t is enabled in M and firing t results in the marking M'. For instance, in the Petri net of Figure 2.2, $(N, [p_0])[t_0\rangle(N, [p_1])$, and $(N, [p_1])[t_1\rangle(N, [p_2, p_3])$.

Let $\sigma = \langle t_1, t_2, \dots, t_n\rangle \in T^*$ be a sequence of transitions. $(N, M)[\sigma\rangle(N, M')$ denotes that there is a set of markings $M_0, M_1, \dots, M_n$ such that $M_0 = M$, $M_n = M'$ and $(N, M_i)[t_{i+1}\rangle(N, M_{i+1})$ for $0 \le i \le n$. A marking M' is *reachable* from M if there exists a σ such that $(N, M)[\sigma\rangle(N, M')$. For instance, for the Petri net of Figure 2.2, $(N, [p_0])[\sigma\rangle(N, [p_{12}])$ with $\sigma = \langle t_1, t_1, t_2, t_4, t_5, t_7, t_9, t_{12}, t_{13}\rangle$.

In order to be able to relate activities to transitions in the Petri net, we define a labeled Petri net in a similar way as in [7]:

Definition 2.12 *(Labeled Petri net)*
A labeled Petri net *is a tuple $N = (P, T, F, A, l)$ where (P, T, F) is a Petri net as defined in Figure 2.2.2, $A \subseteq \mathscr{A}^\tau$ is a set of* activity labels*, and $l : T \rightarrow A$ is a* labeling function*. Let $\sigma' = \langle a_1, a_2, \dots, a_n\rangle \in \mathscr{A}^*$ be a sequence. $(N, M)[\sigma'\rangle(N, M')$ if and only if there is a sequence $\sigma \in T^*$ such that $(N, M)[\sigma\rangle(N, M')$ and $\sigma' = l(\sigma_1) \cdot l(\sigma_2) \cdot \dots \cdot l(\sigma_{|\sigma|})$.*

We define a *completed trace* of a Petri net, and the language that a Petri net can produce, as follows:

Definition 2.13 *(Completed traces, language of a Petri net)*
Let $N = (P, T, F, A, l)$ be a labeled Petri net with initial marking M_{init} and final marking M_{final}. A completed trace *σ_c of a Petri net N is a $\sigma_c \in \mathscr{A}^*$ for which there is a sequence $\sigma \in T^*$ such that $(N, M_{init})[\sigma\rangle(N, M_{final})$ and $\sigma_c = l(\sigma_1) \cdot l(\sigma_2) \cdot \dots \cdot l(\sigma_{|\sigma|})$. The* language *of a Petri net N, $\mathscr{L}(N)$, is defined as all completed traces that N can produce.*

The (labeled) Petri net of Figure 2.2 can produce the completed traces $\langle a, b, c, e, f, h, i, k\rangle$ and $\langle a, b, e, c, f, g, b, c, d, f, h, j, i, k\rangle$. In principle, multiple transitions may have the same label. Some transitions however are unobservable, such as transitions t_{10}, t_{11} and t_{12}. Furthermore, the language described by a Petri net can be infinite. Consider for instance the Petri net of Figure 2.2, which contains a loop. In the remainder of this thesis, with 'Petri net' we actually mean a labeled Petri net.

Although the modeling language of Petri nets is basic, it allows for very expressive descriptions of (concurrent) behavior. Together with its clear semantics, it is the preferred modeling language in business process research [5]. However, it has limitations when used for describing more complicated behavior. As shown in the example process model of Figure 2.2, describing a non-exclusive choice requires additional (silent) transitions and arcs. There are several solutions for modeling non-exclusive choice behavior, however all of them increase the complexity and reduce readability of the process model.

2.2.3 Business Process Model and Notation (BPMN)

A business process modeling notation used extensively in industry is the Business Process Model and Notation (BPMN)[1] [143]. The main purpose of the BPMN language is to create and document process models. In the latest versions executable semantics are provided, enabling execution of the modeled process. The BPMN notation is supported by many tool vendors and has been standardized by the OMG (Object Management Group) [143]. The BPMN notation is rather extensive with different types of (hierarchical) activities, events, gateways and even conversations and choreographies. This results in the full BPMN notation containing many different symbols. Moreover, sometimes it is hard to understand the nuances captured in a process model. Therefore, in this thesis we only use a more commonly used subset of the notation. This subset is simpler and commonly understood. Examples of the full BPMN notation can be found in [142].

The formal semantics of BPMN are complex and problematic for the full language, but well understood for the subset used in this thesis. Like for Petri nets, token-based semantics exist for BPMN.

An example of a BPMN process model is shown in Figure 2.3. BPMN has specific control flow operators, called gateways, to specify how the different process model parts are related. Before task `b`, there is an exclusive choice join, merging the two incoming branches. After activity `b` there is a parallel gateway, activating both outgoing branches. The branch going down reaches an event-based gateway, followed by two event triggers. In case a message (e.g.

[1] Throughout the remainder of this thesis, BPMN refers to BPMN version 2.0

e-mail) comes in, `d` is activated. If after a certain amount of time no message has been received, the time-out is triggered, activating `e`. When activity `c` is also executed, activity `f` is enabled. Next a choice is made whether activity `g` or `h` is activated. Another gateway is shown after task `h`, which is the non-exclusive or *OR* gateway. The behavior is the same as in the Petri net model: tasks `i` and `j` can be executed in parallel and at least one has to be executed before the process can continue. After execution of activity `k` the process is concluded.

Besides the constructs shown in Figure 2.3 others are available. For instance, through so-called swimlanes, it can be indicated which group or role executes a particular activity. Additionally, BPMN proces models may contain information regarding which activities use which documents or data objects, how messages flow and how participants within the process interact. However, research [189] has shown that typically less than 10 different symbols are used, while more than 50 distinct graphical elements are available.

2.3 Event Logs

An event log stores the execution history of a process. Table 2.1 shows an excerpt of an example dataset used for process mining. Our example log stores some execution history of a loan application process (see Section 1.2.1). An event log contains data related to a *single process*. Each line in the table represents one event and each column represents an attribute of this event. An event is associated with a case, or *process instance*. In Table 2.1 the events are already grouped by case and sorted chronologically. The sequence of events that is recorded for a process instance is called a *trace*. The first recorded event in the table is related to case 1 and represents the execution of the activity `Register application` by Pete on December 30, 2010. Additional attributes can be re-

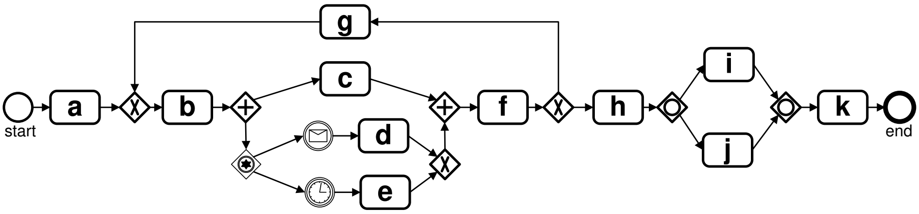


Figure 2.3: Example of a BPMN model.

Table 2.1: Example event data (from [5]).

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010 11:02	Register application	Pete	50	...
	35654424	31-12-2010 10:06	Check credit	Sue	400	...
	35654425	05-01-2011 15:12	Calculate capacity	Mike	100	...
	35654426	06-01-2011 11:18	Check system	Sara	200	...
	35654427	07-01-2011 14:24	Reject request	Pete	200	...
	35654427	08-01-2011 09:03	Send decision e-mail	Pete	200	...
2	35654483	30-12-2010 11:32	Register application	Mike	50	...
	35654485	30-12-2010 12:12	Calculate capacity	Mike	100	...
	35654487	30-12-2010 14:16	Check credit	Pete	400	...
	35654488	05-01-2011 11:22	Accept request	Sara	200	...
	35654489	08-01-2011 12:05	Send decision e-mail	Ellen	200	...
3	35654521	30-12-2010 14:32	Register application	Pete	50	...
	35654522	30-12-2010 15:06	Check system	Mike	400	...
	35654524	30-12-2010 16:34	Check credit	Ellen	100	...
	35654525	06-01-2011 09:18	Calculate capacity	Sara	200	...
	35654526	06-01-2011 12:18	Accept request	Sara	200	...
	35654527	06-01-2011 13:06	Send decision e-mail	Sean	400	...
4	35654641	06-01-2011 15:02	Register application	Pete	50	...
	35654643	07-01-2011 12:06	Check credit	Mike	100	...
	35654644	08-01-2011 14:43	Calculate capacity	Sean	400	...
	35654645	09-01-2011 12:02	Reject request	Sara	200	...
	35654647	12-01-2011 15:44	Send decision e-mail	Ellen	200	...
⋮	⋮	⋮	⋮	⋮	⋮	⋮

lated to this event such as the incurred cost, data attributes entered, etc. Events need to be uniquely identified, which is achieved by assigning unique identifiers. It is important that each event is related to a case and that events are sorted. In general the timestamp of execution is used to sort events chronologically. The resulting sequence of events is referred to as a *trace*. The times shown in Table 2.1 should be interpreted as completion times, i.e., the time when the corresponding activity was completed. In general events can also be recorded to register when activities are started, paused and resumed, etc. For a more complete overview of the different state changes of activities that can be recorded we refer to [5]. In the remainder of this thesis, when we refer to an event, we usually refer to the completion of the corresponding activity. Furthermore, please note that an activity can be executed multiple times for the same case, resulting in different events relating to the same activity and case.

We formalize an event log as follows:

Definition 2.14 *(Trace, Event Log)*
Let $A \subseteq \mathscr{A}$ be a set of activities. A trace *$t \in A^*$ is a sequence of activities. An* event log *$L \in \mathbb{B}(A^*)$ is a multiset of traces.*

The trace for case 1 of Table 2.1 is $\langle$register application, check credit, calculate capacity, check system, reject request, send decision e-mail$\rangle$. For convenience we abbreviate full activity names to single letters, e.g., the trace for case 1 then becomes $\langle a,b,c,d,f,g\rangle$. The event log of Table 1.2a for instance can be represented as $[\langle a,b,c,d,f,g\rangle^{38},\langle a,b,d,c,f,g\rangle^{26},\dots,\langle a,c,b,f,g\rangle]$. In practice, each event and trace contains more information that can be considered. In the remainder of this thesis we refer to traces and events, and only consider a trace to be a sequence of activities.

An event log can be related to a process model, for instance a Petri net, via the activities in the process model and traces of the event log. For instance, we can say that a trace t in an event log L (i.e., $t \in L$) can be replayed in a Petri net N if $t \in \mathscr{L}(N)$. Note however that, even when an event log contains observed behavior that can be related to a specific process, not all traces of the event log have to fit the associated process model perfectly.

The event data shown in Table 2.1 can be easily converted into an *event log*. An event log is a pre-defined structure for storing event data. The de facto standard for storing event logs on disk is the XES [176] event log format. XES stands for *eXtensible Event Stream* and is the successor of the popular MXML [65] event log format. The XES standard stores information regarding the event log as a whole, the traces, and the events belonging to the traces.

Chapter 3

Process Trees

One of the main requirements for process discovery algorithms is to produce correct, also referred to as *sound*, process models, as is mentioned as Challenge 1 in Section 1.3. And although the correctness of a process model can be verified, making an incorrect process model correct is not always possible. Therefore, this chapter introduces process trees as a process model notation that guarantees soundness. Besides soundness, process trees are also easy to translate into various other process modeling notations, thus addressing Challenge 2. Moreover, the translated process models are inherently structured and thus easier to understand.

Section 3.1 discusses several general requirements for process modeling notations in the context of process discovery, of which soundness is the most important requirement. Section 3.2 lists several commonly used process modeling notations. In Section 3.3 we then evaluate to what degree these process modeling notations meet the requirements discussed in Section 3.1. Next we present our process tree notation in Section 3.4. Translations to and from process trees are discussed in Section 3.5 and Section 3.6 concludes this chapter.

3.1 Requirements

When choosing a process modeling notation to use for process discovery, one has to be fully aware of the implications of such a representational bias [4, 6, 11]. Choosing a suitable process model notation for process discovery is a very important design decision. The choice of process model notation has great impact on the (im)possibilities of the algorithm. Therefore, in this section we list the most important requirements that should be considered when choosing which process modeling notation to use.

3.1.1 Soundness and Relaxed Soundness

As mentioned in Section 1.3 as Challenge 1, it is important that processes are error-free, i.e., sound. Soundness is a domain-independent but crucial property that any process model should satisfy [69, 133]. However, many process modeling notations allow for the creation of unsound models [5, 13, 101, 143]. While soundness can be determined for some (subclasses of) process modeling notations [16, 172, 173], for highly expressive modeling notations this is not always possible [15, 16, 67].

Since soundness is defined differently for various notations, we use the definition from [5, 16] for Petri nets, which define soundness as:

1. *Option to complete:* for each possible marking a process can be in, it should always be possible to reach the final marking.

2. *Proper completion:* when the process model reaches its final marking (i.e., is finished), no other work should be left.

3. *No dead transitions:* all parts of the process model are potentially reachable.

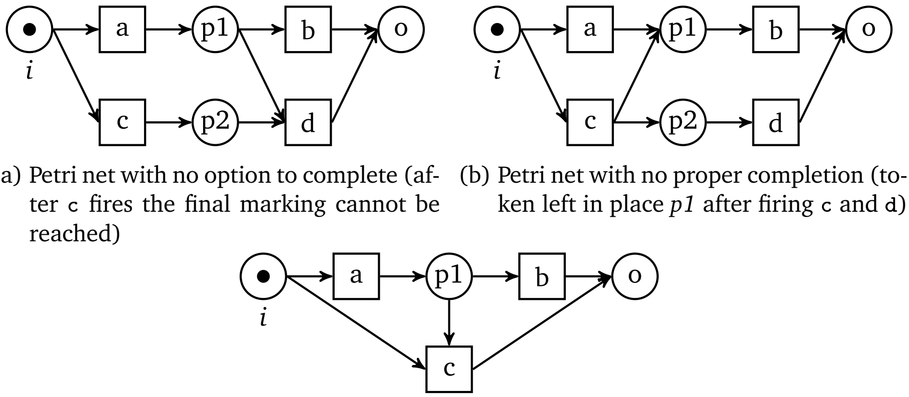
Violating any of these three properties makes a process model unsound. Figure 3.1 shows three Petri nets that each violate a rule. For each of these Petri nets place i is the initial marking ($M_{\text{init}} = [i]$), i.e., the only place that initially contains a token. Furthermore, each of the Petri nets is finished when there is a token in place o and nowhere else, hence $M_{\text{final}} = [o]$ is the final marking. The first Petri net, shown in Figure 3.1a, does not have the option to complete. The final marking $[o]$ cannot be reached once activity c fires, since activity d will never be enabled. In the Petri net shown in Figure 3.1b the proper completion property is violated. The final marking can never be reached since there is a token remaining in place p_1, after transitions c and d have fired, allowing for activities to still be executed. Hence marking $[o]$ is unreachable. The third soundness property is violated in the Petri net shown in Figure 3.1c. In this Petri net transition c can never be executed since there is never a token both in place i and p_1 at the same time.

Whether a process model violates one or more of these requirements cannot always be detected [173]. Moreover, when a violation is detected, repairing the model is often far from trivial. The main problem is that several solutions allowing for different behaviors may be possible, making it hard to choose a solution. Therefore, we propose to prevent the creation of unsound process models.

The notion of *relaxed soundness* [16, 62] is a relaxation of the previous soundness notion. Relaxed soundness states that a Petri net is sound if every transition is on a path from the initial marking to the final marking, and thereby drops the proper completion requirement. Intuitively this means that there exist enough executions which terminate properly, i.e., without violating the proper completion requirement. 'Enough' in this case means that each transition is covered. The relaxed soundness notion is meant to be closer to the intuition of the modeler. However, it allows for execution sequences without proper completion, and therefore is still not desirable since not all analysis algorithms can handle this.

3.1.2 Expressiveness

The second requirement for process model notations is that of expressiveness. The behavior of the main process model notations has been characterized using



(a) Petri net with no option to complete (after c fires the final marking cannot be reached)

(b) Petri net with no proper completion (token left in place *p1* after firing c and d)

(c) Petri net with a dead transition c

Figure 3.1: Examples of unsound Petri nets.

an extensive collection of control flow patterns [17]. Based on these patterns, we can conclude that a process model notation should be able to express the following key aspects of process behavior:

1. *Concurrency* between different parts of the process is one of the essential behavioral aspects that a process modeling notation should cover. Since work is often done by different teams of users, different parts of a case can be executed in parallel. Enumerating all possible interleavings does not result in a readable process model, since the total number of interleavings is exponential in the number of activities, and in case of loops even infinite. Therefore, the process modeling notation should be able to express concurrency between different parts of the process.

2. *Silent actions*, although not related to a concrete activity, often express crucial control-flow aspects such as the possibility to skip an activity. If the chosen notation does not support silent actions, certain control-flow constructs cannot be expressed and hence can not be discovered by the discovery algorithm.

3. *Duplication of activity labels* should be allowed. The same activity can possibly be executed in different parts or states of the process, and therefore it should be possible to model the same activity multiple times within the same process model.

4. *Non-free-choice* behavior in a process, i.e., choices that depend on decisions made earlier in the process, should be captured. If the notation cannot support such long-term dependencies, then a discovery algorithm is not able to correctly express these in a concise way.

5. The *non-exclusive ('OR') control-flow construct* is a higher-level construct that is hard to express in a lower-level notation such as Petri nets. The notation should natively support OR constructs to allow a discovery algorithm to express this construct. Besides native support, the semantics of the OR construct should be clearly defined in all situations. A lot of research has been done on the OR-join semantics for different process model notations [13, 108, 185, 186]. However, only a few robust implementations have been proposed [108, 185]. This makes expressing non-exclusive behavior in a process modeling notation a difficult task, since it could depend on the semantics used whether a process model is sound.

Together, these five requirements cover the basic aspects of control flow expressiveness of process modeling notations.

3.1.3 Understandability

The discovered process model will be interpreted by the process owner or a process modeling specialist. Therefore, the chosen representation of the process model is crucial for the understandability, and thus the usability, of the discovered process model. Seven process modeling guidelines are mentioned in [132]. Although these relate to process modeling, and not process discovery, they do describe crucial aspects of the ease of understanding process models by the end users.

The key aspect is size: bigger models are harder to read than smaller models. This does not only refer to the number of nodes (e.g. tasks, events) but also to the number of routing paths between them. Related to this is hierarchy in a process model [4, 132]. Especially for larger process models, the ability to group activities hierarchically together makes the process model easier to understand for the user. Using the information available in the event log, a process discovery algorithm could be able to automatically infer a hierarchy for the process model. However, if the representational bias does not allow for the grouping of activities into sub-processes, a process discovery algorithm always finds a flat process model.

Other key features to make a process model understandable are the use of only one start and end place (or activity), and avoiding OR routing and other complex control flow constructs.

Our third requirement therefore is that the process modeling notation should have a representation that is easy for humans to understand.

3.1.4 Formal Semantics

When a process model is discovered from an event log the analysis does not end. Other algorithms can perform further analysis based on the discovered process model. In order for these algorithms to understand and correctly interpret a process model, the model has to have clear formal semantics [1, 183]. Although this sounds trivial, for some popular process modeling notations the semantics are complex or incomplete [57, 64, 108, 185]. For instance the exact semantics of an OR-join can be interpreted in different ways resulting in different interpretations of the same modeling construct.

Therefore, the fourth requirement is that the process model should have clear semantics. In case there are multiple ways in which the semantics of a process model can be interpreted, the process discovery algorithm should be clear on which interpretation is to be used.

3.1.5 Suitable for the Process Discovery Algorithm

One should not forget that the chosen process model notation should be suitable for the process discovery approach in mind. The difference between graph-structured and block-structured modeling notations for instance is important [4]. Usually, the chosen representation and the approach of the process discovery algorithm are tightly coupled [28, 180]. Depending on the approach, some (internal) notations make more sense than others. Furthermore, the ability of a process model to express certain behavior is only of use if the process discovery algorithm is able to discover this behavioral construct. A process modeling notation with less expressive power is therefore not a bad idea if the process discovery algorithm is not able to find more complicated behavioral constructs.

Therefore the fifth requirement is that the chosen process modeling notation should be suitable for the process discovery algorithm, and vice versa.

3.2 Common Process Modeling Notations

The importance of modeling business processes is illustrated by the plethora of process modeling notations, sometimes referred to as the "new Tower of Babel" [5]. In Section 2.2 we discussed labeled transition systems, Petri nets, and BPMN. In this section we address some of the other commonly used process modeling notations. Each of the notations has a different level of expressiveness, as is shown by the different example processes provided in this section.

In Section 3.3 we evaluate these common notations based on the requirements discussed in Section 3.1.

3.2.1 Hidden Markov Models

Hidden Markov models are an extension of ordinary Markov processes. An example of a hidden Markov model is shown in Figure 3.2. A hidden Markov model has a set of states, represented by circles, and transition probabilities. Moreover, unlike standard Markov models, in each state an observation is possible, represented by squares, but the state itself remains hidden. Observations have probabilities per state as shown in Figure 3.2, e.g. in state s4 there is a chance of 0.75 of observing activity d, and of 0.25 of observing activity e.

Three fundamental problems have been investigated for hidden Markov models [5, 26]:

1. Given an activity sequence, how to compute the probability of the sequence given a hidden Markov model?
2. Given an activity sequence and a hidden Markov model, how to compute the most likely "hidden path", in the model?
3. Given a set of activity sequences, how to derive a hidden Markov model that maximizes the probability of producing these sequences?

The last problem is related the most to process discovery, but is also the most difficult problem. Although hidden Markov models are versatile and relevant to process mining, there are several complications [5]. First of all, there are many computational challenges due to the time-consuming iterative procedures. Second, current techniques for discovering a hidden Markov model from observed behavior require the number of hidden states to be predefined. Third, the resulting hidden Markov models are typically not very accessible for the end user. Accurate models are typically large and even for small examples the interpretation of the states is difficult. Clearly, hidden Markov models are at a lower abstraction level than modeling notations such as BPMN and Petri nets.

3.2.2 Yet Another Workflow Language (YAWL)

YAWL (which stands for "Yet Another Workflow Language") is an open-source process execution engine with a corresponding process modeling and execution language [101]. The aim of YAWL is to offer comprehensive support for the workflow patterns [17], covering not only the control flow perspective but also

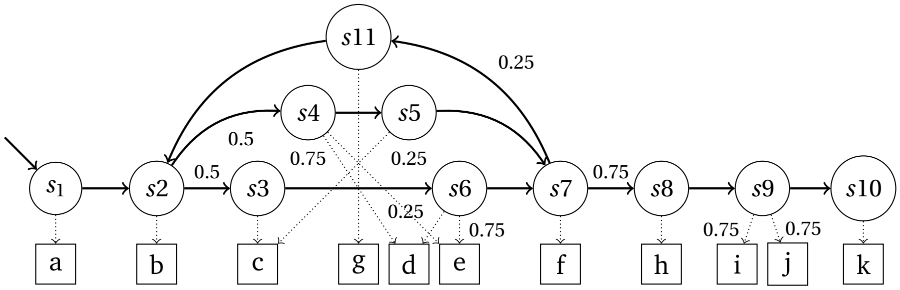


Figure 3.2: An example HMM model, with 11 (hidden) states and 11 possible observations. All arcs shown have weights attached, only weights not equal to 1.0 are shown.

data patterns, resource patterns, and exception patterns, while at the same time keeping the language relatively simple. In this chapter, we restrict ourselves to the control flow perspective.

Figure 3.3 shows an exemplative process in the YAWL notation. The notation of YAWL has been derived from Petri nets. The initial and final states are marked clearly by places filled respectively with a play and stop symbol. Furthermore, for the tasks the split and join semantics can be specified. Task b for instance has XOR-join semantics, as is indicated by the rectangle facing with the tip towards the incoming arrows. At the same time b has AND-split semantics, as is indicated by the rectangle facing with the long edge towards the outgoing arrows. Although the exclusive split and join semantics can be made explicit in the tasks, the deferred choice is also still available by using the conventional Petri net notation, as is shown between tasks d and e. After b is executed, it is also possible to execute g, which cancels the execution in the boxed region of the process model. Task f demonstrates AND-join semantics, and by executing f, g is disabled. Task f is followed by task h which has OR-split semantics, for which the enabled outgoing arcs (at least one) are evaluated using data conditions.

YAWL is both a rich proces modeling language and an open-source workflow system. The YAWL language has a sound underlying formalization, and is based on the workflow patterns [17]. At the same time the YAWL language is kept simple. Moreover, YAWL models are executable in the YAWL workflow system.

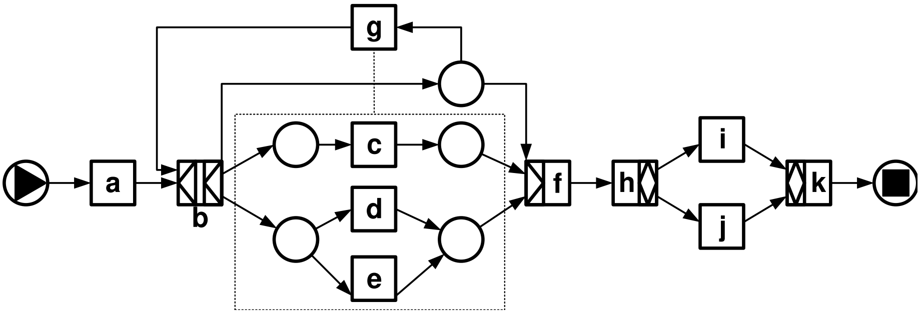


Figure 3.3: An example YAWL process model where g cancels a region.

3.2.3 Event-Driven Process Chains (EPCs)

The Event-Driven Process Chain (EPC) [159] is another graphical process modeling notation, originally introduced in the context of SAP R/3. An example of an EPC model is shown in Figure 3.4. EPCs are based on activities, *functions* in EPCs, and *events*. Both functions and events have exactly one input and one output arc. A notable exception are the start and end events which have only one outgoing or one incoming arc, respectively. Furthermore, functions and events should alternate, i.e., no two events or two functions can be connected either directly or through a path of connectors. Additionally, EPCs have connector nodes, much like the gateways of BPMN. Connector nodes support the control flow constructs of parallelism, exclusive choice and non-exclusive choice. The EPC notation was one of the first notations allowing for non-exclusive (i.e., OR) splits and joins. However, no clear semantics nor reference implementations were provided [13].

EPCs are supported by commercial products such as ARIS and SAP R/3. However, EPCs also have much of the same issues as the other languages we discussed. Since EPCs contain higher-level constructs, as do the BPMN and YAWL languages, all kinds of subtle semantic problems may arise. For instance the execution semantics of the vicious circle, where several OR-connectors wait on each other, is not handled by the EPC definition [108].

3.2.4 Causal Nets

A *Causal net* (or C-net) is a process model notation tailored towards process discovery [9]. All of the process modeling languages described so far connect activities (i.e., transitions (Petri nets), tasks (YAWL and BPMN), and functions (EPC)) through model elements indicating state and/or control flow (i.e., places (Petri nets), conditions, connectors and events (EPC), gateways and events (BPMN)). These elements however do not explicitly leave their footprints in an event log, which is the input for process discovery. A causal net is a graph where nodes represent activities and arcs represent causal dependencies. Each activity has a set of possible *input bindings* and a set of possible *output bindings*. An example of a causal net is shown in Figure 3.5, which describes the same process as the examples of the previous proces modeling notations. Activity `a` has only an empty input binding as this is the start activity. There is only one possible output binding: `{b}`. Activity `b` however has two possible input bindings: `{a}` and `{g}`, hence `b` is preceded by `a` or `g`. After execution of `b` two output bindings can be triggered: `{c,d}` or `{c,e}`, indicating that activity `c` is al-

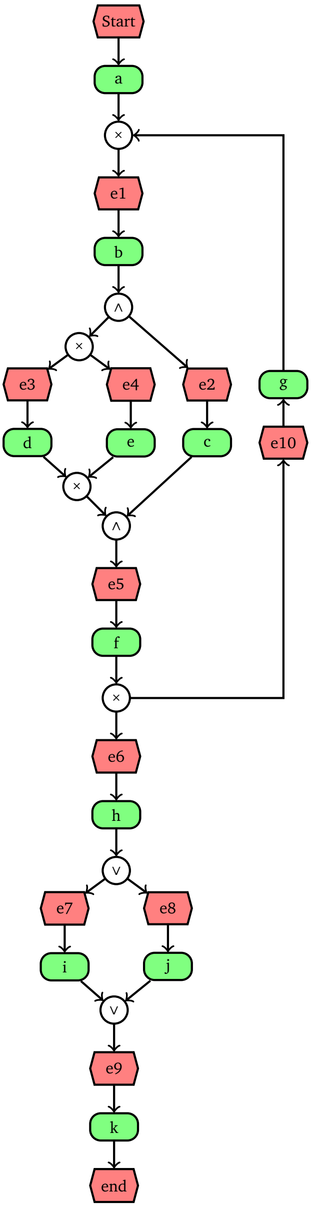


Figure 3.4: An example EPC model.

ways enabled, together with either activity d or e. The non-exclusive split after activity h is expressed by indicating that each of the two exiting branches can be activated alone, or both can be activated. The way causal nets denote activity bindings allows for a concise way of encoding activity relations.

As a downside, causal nets are hard for humans to quickly read and understand. This is mainly caused by the information density, i.e., the control flow is encoded in the arcs, and not by explicit symbols. The main disadvantage however is that causal nets are tailored towards process discovery, and not proces modeling. In essence, this means that a given sequence of activities can be verified if it fits a given causal net. However, causal nets are not intended to provide executionable semantics since they do not fix the moment of choice (they use trace-based semantics). Therefore, causal nets are mainly used as an intermediary process model notation for process discovery algorithms to discover a process model. The causal net is then later translated to a more human-friendly visual representation.

3.2.5 Heuristics Net

A Heuristics net is produced by the heuristics miner [180, 181]. A Heuristics net is in essence another representation of a causal net, where the activity bindings can be visualized as choices in a BPMN notation style. An example of a heuristics net is shown in Figure 3.6. Multiple arcs exiting from an activity are activated simultaneously, and are thus executed in parallel. It should be

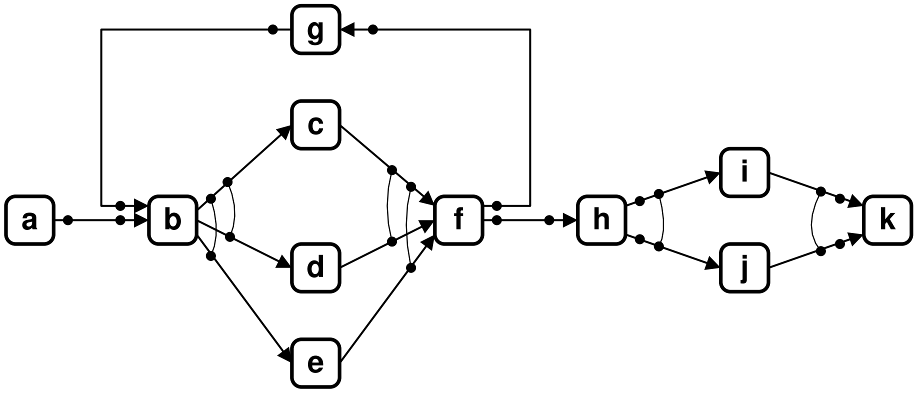


Figure 3.5: An example causal net.

noted however, that some relations are not expressed correctly, such as the OR construct between i and j.

3.2.6 Fuzzy Models

Fuzzy models are the output format used by the fuzzy miner [96, 97]. An example of a fuzzy model, based on the behavior as described by the previous models, is shown in Figure 3.7a. Fuzzy models are called fuzzy because they have no explicit semantics and are only based on heuristic replay semantics. The arcs in a Fuzzy model represent a causal relationship. In Figure 3.7 for instance it is encoded that a is always followed by b. It is also shown that after b, activities c, d and e appear in any order, and among them also causal dependencies exist. However, the exact control-flow construct between these activities remains unclear. The benefit of the Fuzzy models however is that they allow for aggregation of parts of the model. Figure 3.7b shows an instance of an aggregated version of the Fuzzy model of Figure 3.7a, where activities d, e, g, h, i and j are combined together.

3.2.7 Process Algebras

Process algebras (sometimes also referred to as ‘process calculus’) allow for the description and analysis of concurrent systems [32, 84, 135]. Many different process algebra languages exist but in this thesis we mainly consider process algebras on a high level, while we provide examples using the Calculus of Communicating Systems (CCS) process algebra [135]. The main purposes of a process algebra are the description and analysis of processes, and most importantly reasoning about the equivalences between processes. All process algebras have in common that they are able to represent interactions between processes as communication between them. Furthermore, they can describe processes and systems using a small collection of primitives and operators for combining those primitives. Basic operators of process algebras are the parallel composition of

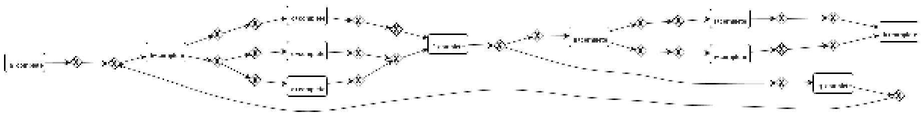
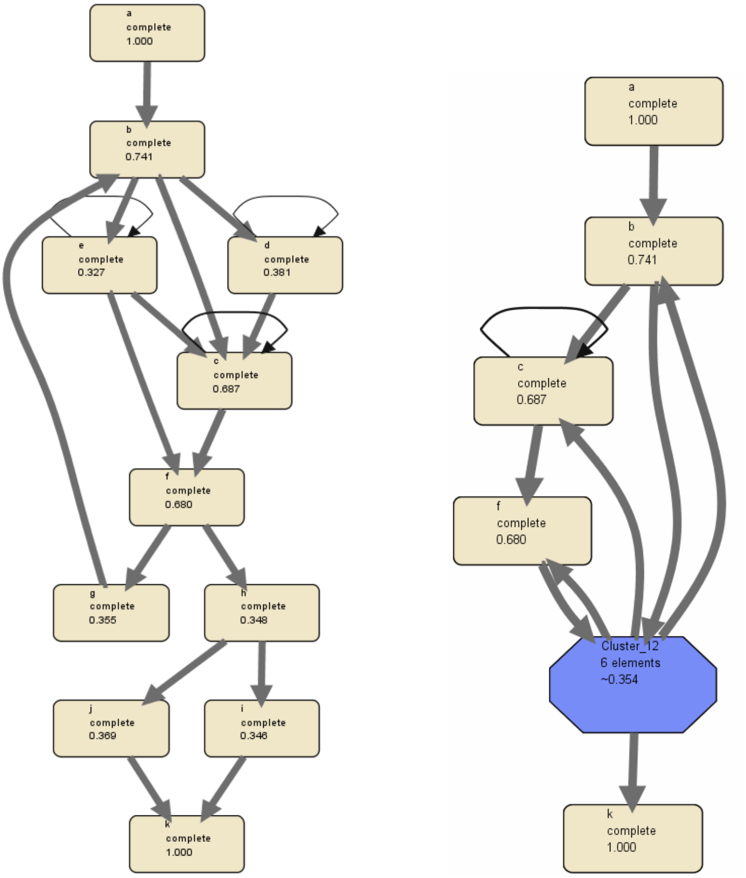


Figure 3.6: An example of a Heuristics net (with split-join semantics shown).



(a) Example of a Fuzzy model without aggregation.

(b) Example of an aggregated Fuzzy model.

Figure 3.7: Example of Fuzzy models describing the same behavior as the previous models.

processes, sequentialization of interactions, hiding of interaction points and recursion of process replication.

In the process algebra CCS [135] the processes as modeled in Figure 3.4 and Figure 3.5 can be defined as follows:

$$
\begin{aligned}
P &::= a.P_1.h.P3.k \\
P1 &::= P2.(\emptyset + g.P1) \\
P2 &::= b.(c|(d+e)).f \\
P3 &::= (i + j + (i|j))
\end{aligned}
$$

Activities are indicated with lower case letters, and a process definition can refer to other process definitions. Activities and processes are put in sequence using '.', in choice using '+' and in parallel using '|'. The main process definition, P, consists of a sequence of activity `a`, process $P1$, activity `h`, process $P3$ and activity `k`. Process $P1$ defines a loop, where first $P2$ is executed, followed by nothing or activity `g` followed by $P1$ again (i.e., a recursive call). Process $P2$ contains a sequence of activities including parallelism between `c` and the choice between `d` and `e`. Process $P3$ describes an OR construct between activities `i` and `j`, where one of them, or both in parallel can be executed.

3.3 Process Modeling Notations versus Requirements

In Table 3.1 the requirements for process modeling notations of Section 3.1 are set out against the process modeling notations most often used in process discovery.

The first requirement of inherent soundness is satisfied by the transition system, hidden Markov model, fuzzy model and process algebra notations. Hidden Markov models and transitions systems are sound under the assumption that every deadlock state is also a final state, which in general is the case. Fuzzy models are inherently sound since they have no semantics and only represent causal dependencies [96]. Process algebras are inherently sound if they do not communicate [84]. Furthermore, since process algebras are block-structured, there cannot be a mismatch between splits and joins. Although the other notations allow for potentially unsound process models, the process discovery algorithm can still prevent this. In Section 6.9 however we show that many process discovery algorithms may return unsound process models.

All common process modeling notations have support for concurrency, a crucial construct in process models. Transition systems and hidden Markov

Table 3.1: Classification of common process modeling notations on process modeling notation requirements.

	Inherently Sound	Concurrency	Silent Actions	Duplicate Actions	Non-Free-Choice	OR construct	Hierarchy	Clear Semantics
Transition Systems [33]	✓	✗	✓	✓	✗	✗	✗	✓
Hidden Markov models [26]	✓	✗	✓	✓	✗	✗	✗	✓
Petri nets [139]	✗	✓	✓	✓	✓	□	□	✓
YAWL [101]	✗	✓	✓	✓	✓	✓	✓	✓
BPMN [143]	✗	✓	✓	✓	✓	✓	✓	✗
EPCs [159]	✗	✓	✓	✓	✓	✓	□	✗
Causal Nets [9]	✗	✓	✓	□	✓	□	✗	□
Fuzzy Models [96]	□	✓	✗	✗	✗	✗	✓	✗
Heuristics Nets [181]	✗	✓	✗	✗	✓	✗	✗	✓
Process Algebras [84]	✓	✓	✓	✓	✗	✗	✓	✓

models do not support concurrency, as they are a low level modeling language. There, one needs to list all interleavings.

Silent actions are supported by all but the heuristics net and fuzzy model. In the case of the fuzzy model silent actions are not required because of the lack of semantics, while the heuristics net has no way of expressing silent actions.

Duplication of activities is not possible in heuristic nets and in fuzzy models. Causal nets support duplication of activities via an extension by Alves de Medeiros [27].

Non-free-choice behavior cannot be expressed by transitions systems, hidden Markov models, causal nets, fuzzy models and process algebras. Since both transition systems and hidden Markov models describe states and transitions between states, a choice is always local. Fuzzy models cannot express non-free-choice behavior because they do not express choices at all, only causal dependencies. And since process algebras do not allow for dependencies between different parts of the formula, they cannot express non-free-choice behavior between different parts of the process.

The OR-construct is not natively supported by transition systems, hidden Markov models, Petri nets, causal nets, the heuristics and fuzzy models, and process algebras. Transition systems and hidden Markov models do not support the OR-construct because they do not support concurrency. Therefore, they explicitly model all possible execution sequences. For Petri nets and causal nets the OR-construct can be expressed by encoding all allowed executions. However, there is no native support for the OR-construct. The heuristics and fuzzy models, as well as process algebras, do not support the OR-construct at all. It should also be noted that for YAWL, BPMN and EPC the OR-join is not specified for all situations [13, 108, 185, 186].

Hierarchy is supported natively by YAWL, BPMN, Fuzzy models, and process algebras. Extensions exist for Petri nets [81, 103] and EPCs [131] to include hierarchy but this is not supported in the basic definition.

The BPMN, EPC and Fuzzy modeling notations have no clear semantics. For the fuzzy model this is a deliberate choice, hence the name ‘fuzzy’. The BPMN and EPC notations however have some semantics that work in easy and straightforward cases, but for more complicated situations different semantics have been proposed [64, 108, 185]. It should be noted that YAWL has clear formal semantics, especially regarding the OR-join.

3.4 The Process Tree Notation

The comparison of different process modeling notations with the requirements for process modeling notations makes it clear that currently there is no perfect process modeling notation. More importantly, very few notations guarantee soundness of the process models. Notable exceptions are the Fuzzy model notation and process algebras. However, Fuzzy models have no clear semantics. Process algebras are block-structured and therefore always sound (if they are not communicating). And although process algebras are suitable for reasoning about processes, they are not the proper way to communicate a proces description to a user. The process tree notation introduced in this section allows for easy reasoning over, and manipulation of, the model. This is a crucial property for the evolutionary process discovery framework we present in this thesis.

Block-structured process models are inherently sound because they require that each control-flow split has a corresponding join of the same type. Furthermore, they do not allow for any dependencies or arcs to enter or exit in between the join and split, so they keep dependencies local. Graph-based notations, such as Petri nets, YAWL and BPMN, can be made block-structured but the graph-based notation does not make this easy.

Several block-structured process modeling languages exist, such as XRL [114], XLang [169], BPEL [144] and Little-JIL [123]. However, all of these languages are targeted at the execution of business processes, and not at the design of these processes. Although this means that they have proper formal (or at least executable) semantics, they are not suitable for process discovery. Within process discovery the aim is mainly to discover the control flow of a process, while the additional information required to be able to actually execute the process cannot be discovered in general. Moreover, the representation of these languages does not always enforce the block-structure, although only block-structured models are valid.

A more suitable basic representation for block-structured processes seems to be a *tree notation*. Since each split has a corresponding join, and nothing entered or escaped in-between, the split and join do not need to be modeled separately. This means that a tree is inherently sound, since split-join mismatches cannot occur. Therefore we use *process trees* to express block-structured processes in a tree structure.

An example process tree and its translation to a Petri net is shown in Figure 3.8. The root node of the process tree of Figure 3.8a is a sequence operator, as is indicated by the →-symbol. It defines that its children are to be executed in a sequence from left to right. Hence, the next child can only start when the previous child is completely finished. The first child of this root node is the activity a. The second child of the root node is the parallel operator, indicated by the $\wedge$-symbol. It allows the execution of activities b and c in any order. The next sibling is a choice, indicated by the $\times$-symbol, between d and e. Finally, activity f is executed. The Petri net shown in Figure 3.8b describes the behavior of the process tree. Both the process tree and the Petri net allow for the following four traces: $\langle a,b,c,d,f\rangle$,$\langle a,b,c,e,f\rangle$,$\langle a,c,b,d,f\rangle$,$\langle a,c,b,e,f\rangle$.

More formally, a process tree can be defined as follows:

Definition 3.1 *(Process Tree)*
Let $A \subseteq \mathscr{A}$ be a finite set of activities. *A process tree (PT) is a tuple $PT = (N, r, m, c)$, where:*

- *N is the non-empty (ordered) set of* nodes *in the process tree, which is partitioned in two sets, N_L for the leaf nodes and N_O for the operator nodes such that $N_L \cup N_O = N$ and $N_L \cap N_O = \emptyset$.*

- *$r \in N$ is the* root node.

- *O is the set of operator types:* $O = \{\rightarrow, \leftarrow, \times, \wedge, \vee, \circlearrowleft\}$.

- *$m : N \rightarrow A \cup O$ is a mapping function, mapping each node to an operator or activity:* $m(n) = \begin{cases} a \in A \cup \{\tau\} & \text{if } n \in N_L \\ o \in O & \text{if } n \in N_O \end{cases}$

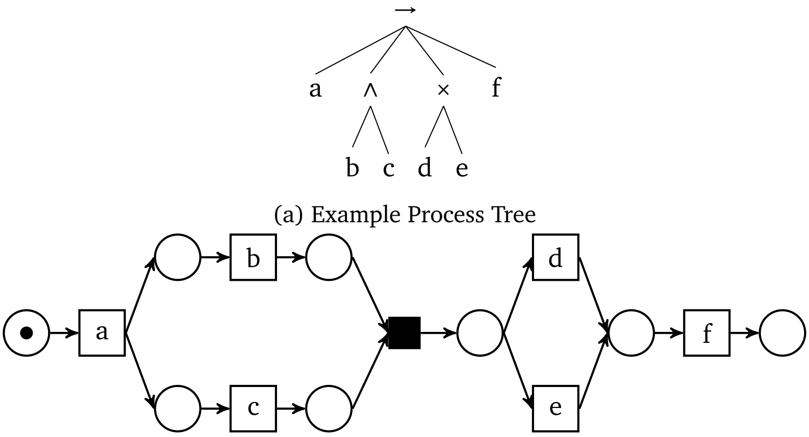
- *$c : N \rightarrow N^*$ is the child-relation function:*
  $c(n) = \langle\rangle \quad$ *if* $n \in N_L$
  $c(n) \in N^+$ *if* $n \in N_O$
  such that

  - *each node except the root node has exactly one parent:*
    $\forall n \in N \backslash \{r\} : \exists p \in N_O : n \in c(p) \wedge \nexists q \in N_O : p \neq q \wedge n \in c(q)$;

  - *the root node has no parent:*
    $\nexists n \in N : r \in c(n)$;

  - *each node appears only once in the list of children of its parent:*
    $\forall n \in N : \forall_{1 \leq i < j \leq |c(n)|} : c(n)_i \neq c(n)_j$;

  - *a node with a loop as operator type has exactly three children:*
    $\forall n \in N : (m(n) = \circlearrowleft) \Rightarrow |c(n)| = 3$.



(a) Example Process Tree

(b) Petri net translation

Figure 3.8: Example process tree and its Petri net translation.

- *$s : N \rightarrow N^*$ is the subtree function, returning all nodes of n in a pre-order:* $s(n) = \begin{cases} n & \text{if } n \in N_L \\ n \cdot s(c(n)_1) \cdot \ldots \cdot s(c(n)_{|c(n)|}) & \text{if } n \in N_O \end{cases}$

- *A tree cannot contain loops:* $\forall n \in N \backslash \{r\} : \exists p \in N_O : n \in c(p) \wedge p \notin s(n)$

- *A node $n \in N$ can be denoted in shorthand as follows: $n = t\langle n_1, \ldots, n_k \rangle$ where $t = m(n)$ and $\langle n_1, \ldots, n_k \rangle = c(n)$.*

Definition 3.1 describes an ordered (rooted) labeled tree structure, where the pre-order relation $\prec$ sorts the nodes in a tree. Moreover, each node has one or more child nodes, except for the leaves.

Process trees have six different operators: sequence (→), the reversed sequence (←), exclusive-choice (×), parallelism (∧), non-exclusive choice (∨) and the loop (⥀). The language of a process tree is defined as follows:

Definition 3.2 *(Process Tree Language)*
Let $A \subseteq \mathscr{A}$ be a set of activities and let $PT = (N, r, m, c)$ be a process tree. The language of a Process Tree $\mathscr{L} : N \rightarrow A^*$ is defined as the language of the root note: $\mathscr{L}(r)$. The language of a node n in a Process Tree is defined as follows:

- *if $m(n) = \tau$, then $\mathscr{L}(n) = \{\langle\rangle\}$*

- *if $m(n) = a \in A$, then $\mathscr{L}(n) = \{\langle a \rangle\}$*

- *if $m(n) \in O$ and $c(n) = \langle n_1, \ldots, n_k \rangle$, then*

  - *if $m(n) = \rightarrow$, then $\mathscr{L}(n) = \{\sigma | \exists_{\sigma_1 \in \mathscr{L}(n_1) \ldots \sigma_k \in \mathscr{L}(n_k)} : \sigma = \sigma_1 \cdot \ldots \cdot \sigma_k\}$*
  - *if $m(n) = \leftarrow$, then $\mathscr{L}(n) = \{\sigma | \exists_{\sigma_1 \in \mathscr{L}(n_1) \ldots \sigma_k \in \mathscr{L}(n_k)} : \sigma = \sigma_k \cdot \ldots \cdot \sigma_1\}$*
  - *if $m(n) = \times$, then $\mathscr{L}(n) = \{\sigma | \exists_{1 \leq i \leq k} : \sigma \in \mathscr{L}(n_i)\} = \bigcup_{1 \leq i \leq k} \mathscr{L}(n_i)$*
  - *if $m(n) = \vee$, then $\mathscr{L}(n) = \{\sigma \in A^* | \sigma = \langle\rangle \Rightarrow (\exists_{n' \in c(n)} : \langle\rangle \in \mathscr{L}(n')) \wedge \sigma \neq \langle\rangle \Rightarrow (\exists_{f:\{1 \ldots |\sigma|\} \rightarrow c(n)} : \forall_{n' \in Rng(f)} : \sigma_{\downarrow n'} \in \mathscr{L}(n'))\}$*
  - *if $m(n) = \wedge$, then $\mathscr{L}(n) = \{\sigma \in A^* | \sigma = \langle\rangle \Rightarrow \forall_{n' \in c(n)} : \langle\rangle \in \mathscr{L}(n') \wedge \sigma \neq \langle\rangle \Rightarrow (\exists_{f:\{1 \ldots |\sigma|\} \rightarrow c(n)} : \forall_{n' \in Rng(f)} : \sigma_{\downarrow n'} \in \mathscr{L}(n') \wedge \forall_{n' \in c(n) \backslash Rng(f)} : \langle\rangle \in \mathscr{L}(n'))\}$*
  - *if $m(n) = ⥀$, then $\mathscr{L}(n) = \{\sigma_1 \cdot \sigma_2 \cdot \sigma_3 \in A^* | \sigma_1 \in \mathscr{L}(c(n)_1) \wedge \sigma_3 \in \mathscr{L}(c(n)_3) \wedge \sigma_2 \in f(c(n)_2, c(n)_1)\}$ with $f : N \times N \rightarrow A^* : f(n_1, n_2) = \{\sigma | \sigma = \langle\rangle \vee (\sigma = \sigma_1 \cdot \sigma_2 \cdot \sigma_3 \in A^* \wedge \sigma_1 \in \mathscr{L}(n_1) \wedge \sigma_2 \in \mathscr{L}(n_2) \wedge \sigma_3 \in f(n_1, n_2))\}$*

Consider for instance the process tree of Figure 3.8a which can be written in shorthand as $\rightarrow\langle a, \wedge\langle b,c\rangle, \times\langle d,e\rangle, f\rangle$. The language of the node $\wedge\ \langle b,c\rangle$ is $\{\langle b,c,\rangle, \langle c,b\rangle\}$, and the language of $\times\ \langle d,e\rangle$ is $\{\langle d\rangle, \langle e\rangle\}$. The language of the whole tree is $\{\langle a,b,c,d,f\rangle, \langle a,c,b,d,f\rangle, \langle a,b,c,d,e\rangle, \langle a,c,b,d,e\rangle\}$. Furthermore, the language of the process tree described by $\vee\langle a,b\rangle$ is $\{\langle a,b\rangle, \langle b,a\rangle, \langle a\rangle, \langle b\rangle\}$. The language of the process tree with a loop, $\circlearrowleft\langle a,b,c\rangle$ is $\{\langle a,c\rangle, \langle a,b,a,c\rangle, \langle a,b,a,b,a,c\rangle, \ldots\}$, which is infinite.

Figure 3.9 shows a slightly more complicated process tree which contains all operators, except the ←-operator. This process tree can be written in shorthand as $\circlearrowleft\langle a, \times\langle \rightarrow\langle b,c,\vee\langle d,e,\rangle\rangle, \wedge\langle f,g\rangle\rangle, h\rangle$. The language of this process tree, $\mathscr{L}(n_0)$, can be defined as follows:

- $\mathscr{L}(n_7) = \{\langle d\rangle\}$.
- $\mathscr{L}(n_8) = \{\langle e\rangle\}$.
- $\mathscr{L}(n_6) = \{\langle d\rangle, \langle e\rangle, \langle d,e\rangle, \langle e,d\rangle\}$ with for
  - $\langle d\rangle$, f is defined as $f(1) = n_7$ and $\langle d\rangle_{\downarrow n_7} = \langle d\rangle \in \mathscr{L}(n_7)$;
  - $\langle e\rangle$, f is defined as $f(1) = n_8$ and $\langle e\rangle_{\downarrow n_8} = \langle e\rangle \in \mathscr{L}(n_8)$;
  - $\langle d,e\rangle$, f is defined as $f(1) = n_7$ and $f(2) = n_8$ and $\langle d,e\rangle_{\downarrow n_7} = \langle d\rangle \in \mathscr{L}(n_7)$ and $\langle d,e\rangle_{\downarrow n_8} = \langle e\rangle \in \mathscr{L}(n_8)$;

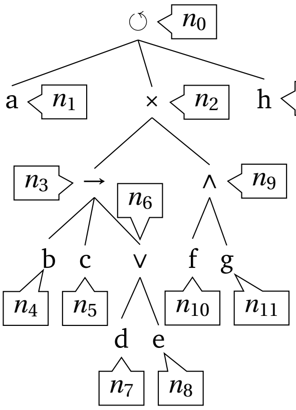


Figure 3.9: Example process tree with for each node the associated index.

  - $\langle e,d\rangle$, f is defined as $f(1)=n_8$ and $f(2)=n_7$ and $\langle e,d\rangle_{\downarrow n_7}=\langle d\rangle \in \mathscr{L}(n_7)$ and $\langle e,d\rangle_{\downarrow n_8}=\langle e\rangle \in \mathscr{L}(n_8)$;.

- $\mathscr{L}(n_4)=\{\langle b\rangle\}$.

- $\mathscr{L}(n_5)=\{\langle c\rangle\}$.

- $\mathscr{L}(n_3)=\{\langle b,c,d\rangle,\langle b,c,e\rangle,\langle b,c,d,e\rangle,\langle b,c,e,d\rangle\}=\langle \sigma_4,\sigma_5,\sigma_6\rangle$ with $\sigma_4\in\mathscr{L}(n_4)$, $\sigma_5\in\mathscr{L}(n_5)$ and $\sigma_6\in\mathscr{L}(n_6)$.

- $\mathscr{L}(n_{10})=\{\langle f\rangle\}$.

- $\mathscr{L}(n_{11})=\{\langle g\rangle\}$.

- $\mathscr{L}(n_9)=\{\langle f,g\rangle,\langle g,f\rangle\}$ with for

  - $\langle f,g\rangle$, f is defined as $f(1)=n_{10}$ and $f(2)=n_{11}$ and $\langle f,g\rangle_{\downarrow n_{10}}=\langle f\rangle \in \mathscr{L}(n_{10})$ and $\langle f,g\rangle_{\downarrow n_{11}}=\langle g\rangle \in \mathscr{L}(n_{11})$;
  - $\langle g,f\rangle$, f is defined as $f(1)=n_{11}$ and $f(2)=n_{10}$ and $\langle g,f\rangle_{\downarrow n_{10}}=\langle f\rangle \in \mathscr{L}(n_{10})$ and $\langle g,f\rangle_{\downarrow n_{11}}=\langle g\rangle \in \mathscr{L}(n_{11})$.

- $\mathscr{L}(n_2)=\{\langle b,c,d\rangle,\langle b,c,e\rangle,\langle b,c,d,e\rangle,\langle b,c,e,d\rangle,\langle f,g\rangle,\langle g,f\rangle\}=\sigma_3\vee\sigma_9$ with $\sigma_3\in\mathscr{L}(n_3)$ and $\sigma_9\in\mathscr{L}(n_9)$.

- $\mathscr{L}(n_1)=\{\langle a\rangle\}$.

- $\mathscr{L}(n_{12})=\{\langle h\rangle\}$.

- $\mathscr{L}(n_0)=\{\langle a,h\rangle,\langle a,b,c,d,h\rangle,\langle a,b,c,e,h\rangle,\langle a,b,c,d,e,h\rangle,\langle a,b,c,e,d,h\rangle,$ $\langle a,f,g,h\rangle,\langle a,g,f,h\rangle,\langle a,b,c,d,b,c,d,h\rangle,\langle a,b,c,d,b,c,e,h\rangle,$ $\langle a,b,c,d,b,c,e,d,h\rangle,\langle a,b,c,d,f,g,h\rangle,\ldots\}=\langle\sigma_1,\sigma_f,\sigma_{12}\rangle$ with $\sigma_1\in\mathscr{L}(n_1)$, $\sigma_2\in\mathscr{L}(n_2)$, $\sigma_f\in f(n_2,n_1)$ and $\sigma_{12}\in\mathscr{L}(n_{12})$ and function f is defined as $f(n_2,n_1)=\{\sigma|\langle\rangle\cup\langle\sigma_2,\sigma_1,f(n_2,n_1)\rangle\}$.

Although soundness is the main motivation for our choice of process trees, the other requirements are also important. The requirement of expressiveness is covered since process trees allow for concurrency (using the $\wedge$ and $\vee$ operators), and silent actions since τ is a possible leaf and duplication of activity labels is allowed. Moreover, the non-exclusive choice control-flow construct is explicitly supported. The only expressive power process trees do not possess is non-free-choice behavior. Because of the block structure, creating long-term dependencies is not possible. However, since process trees allow for the duplication of activities, long term behavior can be explicitly modeled using an ×-operator and duplication of activities.

Besides guaranteed soundness, the block structure also provides a natural hierarchy within the process model. Since operator nodes are allowed to have children with the same operator type, hierarchy can be introduced into the process model. This allows for better readable process models [132].

Although the process tree notation itself provides a clear description of the modeled behavior, it might not be the best way to present the process model to the end user. Therefore we propose to translate the process tree to the process modeling notation preferred by the user. As we show in Section 3.5.1 a process tree can easily be translated to other process modeling notations with clear semantics. Additionally, the hierarchy of the process tree can be maintained, for instance by creating subprocesses if that is supported by the target process model notation.

Furthermore, new operator types can easily be added to the process tree notation, thus allowing for the support of additional control-flow patterns. An example of a type of operator that could be added is the deferred choice. This is done in the CoSeNet notation [161], where the deferred choice and deferred loop variants are added as operator types. However, in this thesis we are only concerned with the language described by the process tree, and therefore can abstract from the moment of choice.

Next to meeting all but one of the process modeling notation requirements, process trees have additional advantages over other process modeling notations:

Reduction of Search Space

Since process trees can only express sound process models, the search space of process trees does not include unsound process models and is therefore smaller than that of graph-based process modeling notations. This makes the notation suitable for search algorithms, such as evolutionary algorithms. Moreover, no valuable time has to be spent on soundness verification and attempts for correction.

Ease of Reasoning

Tree structures are very easy to interpret and modify for algorithms. This facilitates reasoning over process trees. In a process tree for instance it is easy to verify whether two activities or parts of the process are mutually exclusive or in a parallel construct. Moreover, process trees are easy to modify for algorithms because only a few simple checks need to be performed to ensure the consistency of the resulting model.

By choosing process trees as our internal process model notation, we ensure soundness by design. Moreover, process trees have clear formal semantics and have rich expressive power. The only price to pay is the inability to express long-term behavior, caused by the block structure of the process tree. In the next section we translate process trees to the most used process modeling notations in process discovery and vice versa.

3.5 Translations

In this section we show translations from the process tree notation to commonly used process modeling notations. We also discuss how graph-based process modeling notations such as Petri nets can be translated to the process tree notation, using the blocks presented in the translation to that language.

3.5.1 From Process Trees to Other Notations

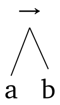
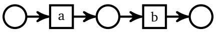
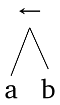
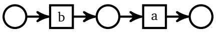
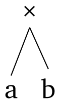
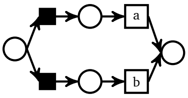
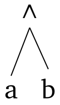
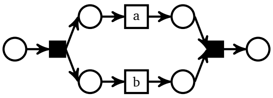
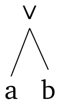
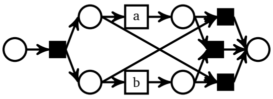
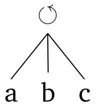
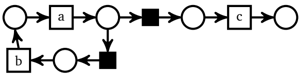
Translating process trees to other process modeling notations is easy because of the clear semantics of process trees. In this section we show translations from the process tree control flow constructs to well-known process modeling notations. Although process trees have some control flow constructs that might not be present in the target process modeling notation, a translation is made so that the same behavior is described. Strong equivalence notions such as (branching) bisimilarity [88] are not always preserved. However, these more strict equivalence notions are not necessary since we are in the domain of process discovery where only the resulting behavior recorded in the event log plays a central role. Therefore, we consider two process models equal when their described languages are equal.

To Petri Nets

The translations of the different process tree operators to Petri net fragments describing the same behavior are shown in Table 3.2. For each of the process tree control-flow constructs a binary process tree part is shown with its corresponding translation to a Petri net. Extending the translation to more than two children is trivial, and is shown for the translations to other process modeling notations. The main reason this is harder to do for Petri nets is the fact that Petri nets do not have a dedicated OR-operator. Correctly expressing the behavior of the OR-operator requires additional places and silent transitions. The provided translation is only one of several possible trace-equivalent alternatives.

Composing the Petri net blocks for process trees with multiple operators can also be done in a straightforward way. Transitions in the Petri net translation can be replaced by the Petri net translation of the sub-tree, where the places are merged. This is also the reason for the explicit silent transitions in the XOR and Loop constructs. Consider for instance the composition of an XOR with a direct loop child. If the silent transitions would not be present in the XOR translation, it would be possible for the loop to execute its ‘do’ and ‘redo’ parts, after which another child of the XOR operator could be executed. The silent transitions in the loop translation are added for the same reason.

Table 3.2: Process tree operators with two or three children (a, b and c), their Petri net translation and their allowed traces.

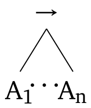
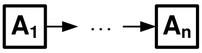
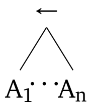
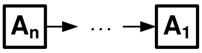
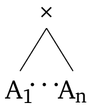
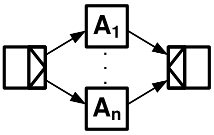
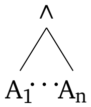
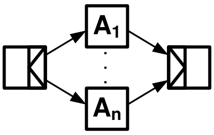
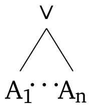
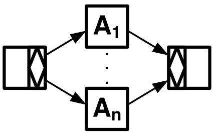
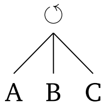
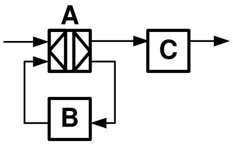
Process Tree	Petri net	Example traces
		1 trace: $\{\langle a, b\rangle\}$
		1 trace: $\{\langle b, a\rangle\}$
		2 traces: $\{\langle a\rangle, \langle b\rangle\}$
		All interleavings of the children: $\{\langle a, b\rangle, \langle b, a\rangle\}$
		All interleavings, with the option to skip all but one child: $\{\langle a\rangle, \langle b\rangle, \langle a, b\rangle, \langle b, a\rangle\}$
		Infinite number of traces of the pattern $a(ba)^*c$: $\{\langle a, c\rangle, \langle a, b, a, c\rangle,$ $\langle a, b, a, b, a, c\rangle, \ldots\}$

To YAWL

The translations of the control flow constructs of process trees to YAWL are shown in Table 3.3. Here the translations are shown for an arbitrary number of children for each operator. Moreover, the children now represent subprocesses and not only activities. The first thing to emphasize is the easy translation of the OR control-flow construct, because of the native support in the YAWL language. Additionally it should be noted that the translation of the XOR construct can be done in different ways in YAWL. We chose to use the explicit XOR choice operator, but this could also have been done by use of the deferred choice, as in Petri nets. In our case there is no difference since we consider only trace equivalence. Similar reasoning holds for the choice in the loop translation.

The different translations can be combined by replacing a transition in a translation with its sub-tree translation. Since YAWL does not enforce the strict alternation between places and transitions, not all operator translations shown in Table 3.3 start or end with a place. The start and end places of two constructed blocks should still be merged, but if a block starts or ends with a transition, these can just be connected with another block without an intermediate place.

Table 3.3: Process tree operators, their translations to YAWL and examples of allowed traces. The capitals A_1, A_n, B and C represent subprocesses.

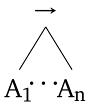
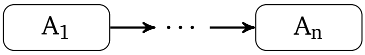
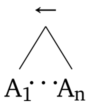
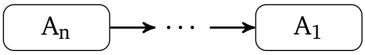
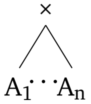
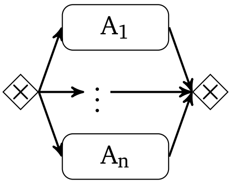
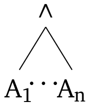
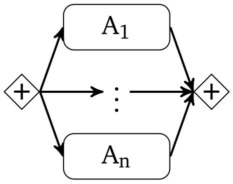
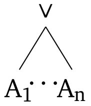
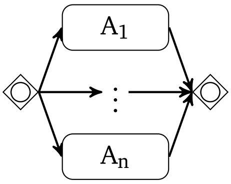
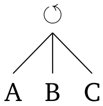
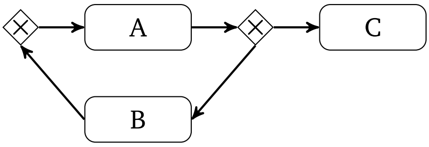
Process Tree	YAWL	Example traces
		$\{\langle A_1, \ldots, A_n \rangle\}$
		$\{\langle A_n, \ldots, A_1 \rangle\}$
		$\{\langle A_1 \rangle, \ldots, \langle A_n \rangle\}$
		$\{\langle A_1, A_n \rangle,$ $\langle A_{1_1}, A_{n_1}, \ldots A_{1_x}, A_{n_y} \rangle,$ $\ldots, \langle A_n, A_1 \rangle\}$
		$\{\langle A_1 \rangle, \ldots, \langle A_n \rangle, \langle A_1, A_2 \rangle, \ldots$ $\langle A_1, A_n, \ldots, A_1 \rangle, \ldots\}$
		$\{\langle A, C \rangle, \langle A, B, A, C \rangle,$ $\langle A, B, A, B, A, C \rangle, \ldots\}$

To BPMN

Translating the different process tree constructs to the BPMN process modeling notation is again straightforward, as is shown in Table 3.4. Since BPMN has built-in support for the OR control-flow construct the translation is straightforward. Again, as for the YAWL translation, the exclusive choice can be translated in two ways, where we chose the explicit choice gateway.

The different BPMN translations can be combined by simply replacing the BPMN task nodes with the corresponding block for that sub-tree. No nodes should be merged, to prevent unintended and incorrect behavior.

Table 3.4: Process tree operators and their translations to BPMN parts. The capitals A_1, A_n, B and C represent subprocesses.

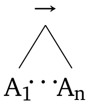
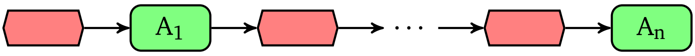
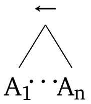
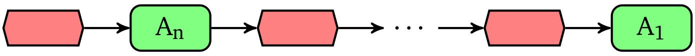
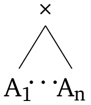
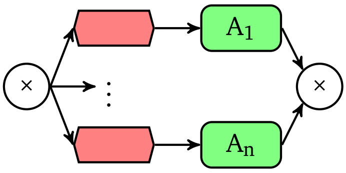
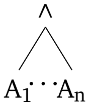
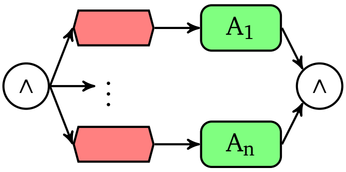
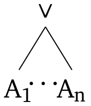
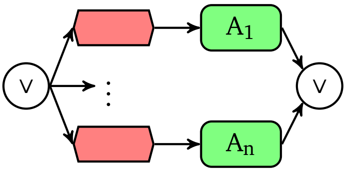
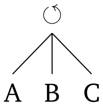
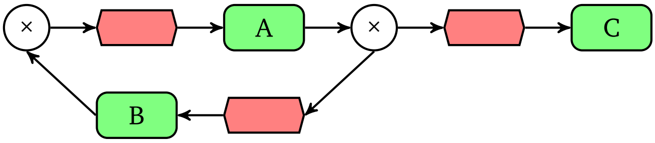
Process Tree	BPMN
	
	
	
	
	
	

To EPCs

The translation of the different process trees constructs to EPCs is shown in Table 3.5. Each of the control flow operators used in process trees has a straightforward translation to an EPC fragment. The loop is again translated using exclusive choice operators.

Translating a full process tree instance to an EPC process model can be done by glueing the EPC fragments together. The functions can be replaced by the corresponding EPC fragment. Since the requirement for EPCs is that events and functions alternate, events should be mapped onto each other when building the final EPC. Functions however should not be mapped. Additionally, since the EPC should start and end with an event, an end event should be added at the end of the translated EPC.

Table 3.5: Process tree operators and their EPC translations (for convenience drawn horizontally). The capitals A_1, A_n, B and C represent subprocesses.

Process Tree	EPC
	
	
	
	
	
	

To Process Algebra (CCS)

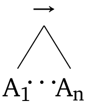
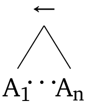
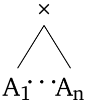
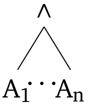
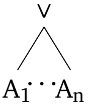
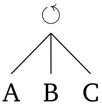
Process trees are closely related to process algebras; translating a process tree to a process algebra is therefore straightforward and is shown in Table 3.6 for the CCS process algebra language. Most process tree operators can be translated by using the corresponding CCS operators. The $\circlearrowleft$-operator of process trees can be translated to CCS by defining two processes to represent the possibly infinite looping behavior. The $\vee$-operator cannot be translated to CCS directly and all possible combinations of the children of the $\vee$-operator need to be encoded in CCS.

To CoSeNet

Within the CoSeLoG project another process tree notation has been developed. In this thesis however, we use a subset of the capabilities of the CoSeLoG project's more extensive process tree notation. Since we are only considering process discovery, constructs such as deferred choice are not required. Furthermore, the CoSeLoG process trees allow for sharing of subtrees, which is not directly supported in our situation since we need to attach information to each node on how it relates to the event log (see Section 5.4 and Section 6.3). Since each subtree, even though they are identical, is positioned in a different part of the process tree, the related behavior is different and therefore these identical subtrees need to be modeled multiple times.

Since our process trees only cover a subset, no additional translation steps are necessary.

Table 3.6: Process tree operators and their CCS translation. The capitals A_1, A_n, B and C represent subprocesses.

Process Tree	**CCS**
	$P ::= A_1.\cdots.A_n$
	$P ::= A_n.\cdots.A_1$
	$P ::= A_n + \cdots + A_1$
	$P ::= A_1 \vert \cdots \vert A_n$
	$P ::= (A_1 + \cdots + A_n + (A_1 \vert A_2) + \cdots + (A_1 \vert A_n) +$ $(A_1 \vert A_2 \vert A_n) + \cdots + (A_1 \vert A_{n-1} \vert A_n) + \cdots + (A_1 \vert \cdots \vert A_n))$
	$P ::= A.P_1.C$ with $P_1 ::= \emptyset + B.A.P1$

3.5.2 From other Notations to Process Trees

Translating a graph-based process model to the process tree notation is not as easy as the translations as discussed in the previous section. Research has been done on transforming graph-based process models to block-structured process models, which are also known as well-structured models [145, 148, 170]. The proposed approaches work by finding single entry, single exit (SeSe) parts of the process model. If parts of the process model are not in a SeSe structure, several approaches [148, 149] can translate the process model, in certain circumstances, to a SeSe structure. A description of a SeSe process model can be created by building an RPST (Refined Process Structure Tree) [170]. However, parts of the RPST might still be unstructured process model sections, captured in a *rigid* section, without the SeSe characteristic. The rigid sections can be classified to define why the given process model is unstructured, but not all classes can be made structured [148, 149].

If the given graph model itself is block-structured, a process tree translation can be made by following the translations of the previous section in the reverse order. It should be noted that our process trees do not distinguish between deferred and explicit choice. Hence, when translating this construct, it must be reduced to the simple ×-operator. This also holds when translating loops.

However, in the case where the graph model is not block-structured, it has to be made block-structured before it can be translated to a process tree [148, 149]. Since we mainly focus on the language of the process model, in the worst case scenario we can simply write out the possible behavior of the process model. A common way to do this is as a choice between all possible sequences of activities in that part of the graph-model. Of course, where possible, this should be generalized to the more general control-flow constructs known by process trees. Therefore, each graph-based process model that is not block-structured can be translated into a trace equivalent process tree.

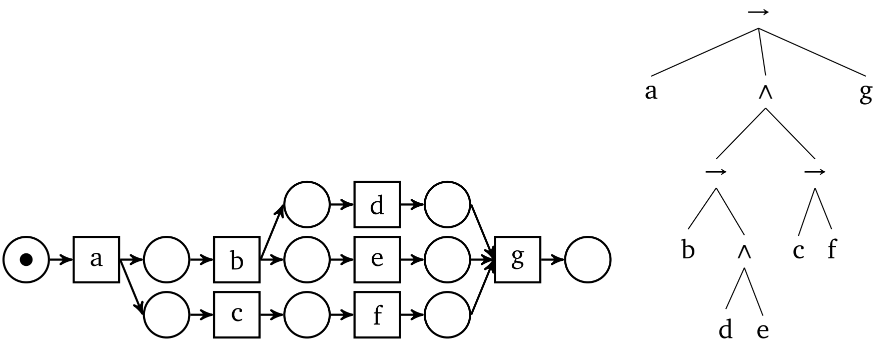
We distinguish three different types of graph-based process models: non-block-structured process models that can trivially be made block-structured, sound process models that are not block-structured, and unsound process models that are not block-structured. Next we show how each of these process model classes could be translated to a process tree equivalent. It is important to note that the latter two types of process models cannot always be made block-structured, as is discussed in [148, 149]. Since structuring process models is a research field in itself, we only present examples of translations, without going into all possible cases and details. For a more detailed discussion we refer to [148, 149]. Additionally we discuss in this section how a CoSeNet can be translated to our process tree notation.

Process models that can be trivially made block-structured

Figure 3.10a shows a process model that is not block-structured but can easily be made so. The process model is not block-structured because the number of split and join activities is not equal. The process model can be made block-structured by adding a silent AND-join transition that joins the places after activities d and e, and outputs to a place that is input to g. After applying this small transformation, which does not change the behavior of the process model, the resulting process model can be easily translated to the process tree as shown in Figure 3.10b.

Sound process models that are not block-structured

Figure 3.11a shows a process model that is not block-structured but can be made so by duplicating activity d. This Petri net has a long term dependency between activities b and e and between c and f. Since process trees cannot capture this, activity d is duplicated in the process tree. This results in the process tree as shown in Figure 3.11b. The resulting process tree has exactly



(a) Petri net which is not block-structured but can easily be made so

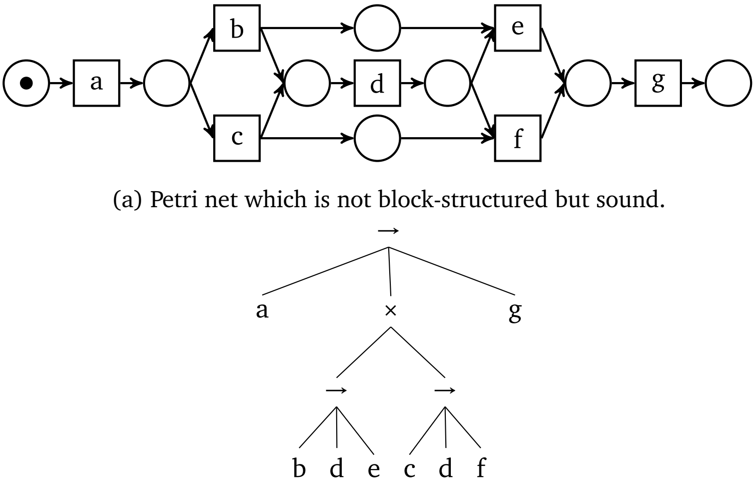
(b) Process tree translation of this process model

Figure 3.10: Process model that can easily be made block-structured, and its translation to a process tree.

the same behavior as the original process model.

Unsound process models

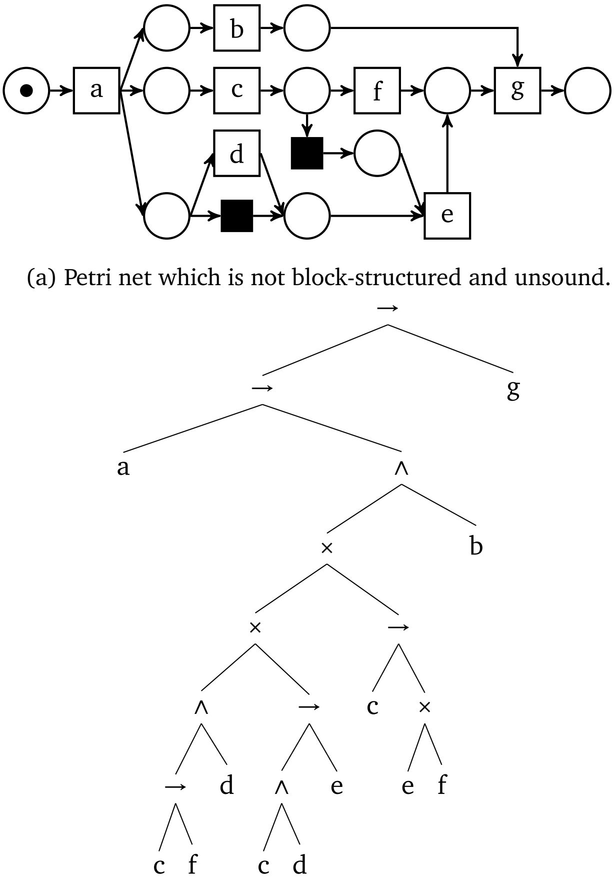
Figure 3.12a shows an unsound process model, which cannot be directly translated to a process tree. The process model is unsound because after executing $\langle a, b, c, d, f, g \rangle$ a token is remaining in the place before activity e while there is also a token in the last place. Because of this improper completion the process model is unsound, as discussed in Section 3.1.1. However, a process tree can still be created, by looking at the traces that can be produced by the Petri net. This process tree is shown below to this Petri net in Figure 3.12b. Although it is not a simple process tree, because several possible traces need to be explicitly encoded, it is a valid translation of the *intended behavior* of the unsound Petri net since the process tree describes the same set of possible traces as the original Petri net.



(a) Petri net which is not block-structured but sound.

(b) Process tree translation of this process model.

Figure 3.11: Process model that is not block-structured but sound, and its translation to a process tree.



(a) Petri net which is not block-structured and unsound.

(b) Process tree translation of the unsound process model.

Figure 3.12: Process model that is not block-structured and unsound, and its translation to a process tree.

From CoSeNet to process tree

As mentioned in Section 3.5.1, within the CoSeLoG project another notation similar to process trees, called CoSeNet, has been developed. Since the CoSeNet notation is more extensive than the process trees used in this thesis, during the translation of a CoSeNet to our notation some additional translation is required. Since we are only considering process discovery, constructs such as deferred choice are not required for our process trees. The main difference with process trees is that in the CoSeNet notation there is a distinction between exclusive and deferred choice, which we both translate to the ×-operator in our process trees. In a similar way the two types of loop known in the CoSeNet notation (one with a deferred choice and one with an exclusive choice to redo the loop) are both translated to our ↺-operator. Furthermore, the CoSeNet notation allows for sharing of subtrees, which is not directly supported in our situation since we need to attach information to each node on how it relates to the event log. Since each subtree, even though they are identical, is positioned in a different part of the process tree, the related behavior is different and therefore these identical subtrees need to be modeled multiple times. Therefore we duplicate shared parts during the translation.

An example of a CoSeNet and the corresponding process tree is shown in Figure 3.13.

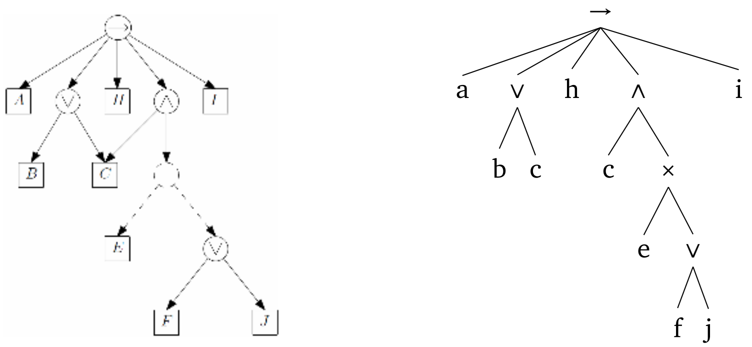


Figure 3.13: Example CoSeNet from [161] and the corresponding Process Tree translation.

3.6 Conclusion

In this chapter we provided solutions to two of the challenges introduced in Section 1.3. Challenge 1, guaranteeing sound process models, is solved by the introduction of process trees. Additionally, this chapter also presented an overview of more requirements for process modeling languages, specifically for the use in process discovery. Existing process modeling notations were evaluated using these requirements. This showed that most notations only fulfill a few requirements. The most important observation is that most process models used for process discovery allow for unsound constructs in the model. Process trees, as defined in this chapter, adhere to most of the requirements discussed. Challenge 2 discussed the separation between visualization and representational bias. Translations from process trees to the different existing process modeling notations ensure that a suitable visualization can be created. We also showed that translating the intended behavior of a process model to a process tree is always possible. In the next chapters we introduce our process discovery algorithm which uses the process tree notation to discover process models.

Chapter 4

A Framework for Evolutionary Process Mining

As discussed in Section 1.3, in process discovery it is important to balance the quality of the discovered process models. In Section 1.3.9 is is argued why evolutionary algorithms are suitable, since they are capable of producing results while aiming for one or more quality dimensions. Therefore, in this chapter we introduce our evolutionary process mining framework called the 'Evolutionary Tree Miner', or ETM for short. First, we describe the overall structure and basic elements of our framework in Section 4.1. We then describe use-cases where this framework can be applied in Section 4.2. Section 4.3 then discusses general requirements an implementation of this framework should adhere to. We then discuss common approaches to the implementation of the different elements of the framework in Section 4.4. Section 4.5 concludes this chapter.

4.1 The ETM Framework

A structured approach for solving a particular class of problems is not always known. In these situations evolutionary algorithms [77] can be applied. Evolutionary algorithms are a subclass of random search algorithms, that create, evaluate and change candidate solutions. These algorithms are inspired by the natural evolution of species. The benefit of evolutionary algorithms is that no structured approach for solving a problem needs to be provided. The most important part of an evolutionary algorithm is how it should evaluate candidate solutions. Other aspects to be specified are the representation of candidate solutions, and the definition of change operations on them. By applying some form of selection, only the better candidate solutions survive to evolve further in later generations. This can result in new and unexpected results, that can be significantly superior to what would have been discovered using a structured approach [102, 107].

The basic flow of an evolutionary algorithm is shown in Figure 4.1. An evolutionary algorithm in general evolves a population of solution candidates over different generations. Using one or more evaluation metrics, the quality of each candidate is calculated. By smartly selecting and changing candidates the quality of candidates can be improved. The ETM framework follows these generic evolutionary steps and uses process trees (see Chapter 3) as the internal representation. Another distinguishing factor of the ETM framework is that evaluation is always done using the four quality dimensions used in process discovery (see Figure 1.2). This ensures that the process information as stored in the event log is closely considered during discovery. Furthermore, the ETM framework allows to use other information sources during discovery.

Technically speaking the ETM framework can be considered an implementation of a *genetic programming* algorithm. Genetic programming algorithms distinguish themselves by the fact that they often use tree(-like) structures to represent candidates in the population. Moreover, they evolve a solution or description of a problem, instead of the optimal parameters for another algorithm to solve the problem. Since the ETM framework evolves process trees, it could be classified as a genetic programming algorithm. However, we refer to the ETM framework as an evolutionary algorithm for sake of simplicity and generality.

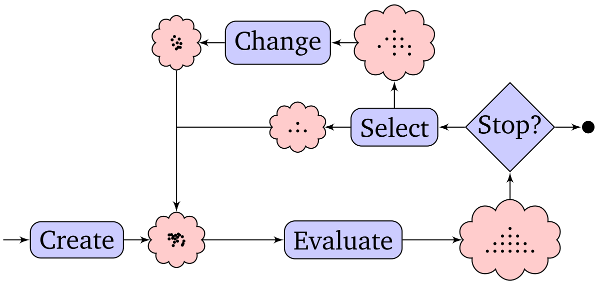


Figure 4.1: The basic framework for evolutionary process discovery.

In the first step of the ETM framework an *initial population* of candidate solutions is created. Different techniques exist to construct random tree structures [129] (see Section 6.1.1).

In the next step each of the candidate solutions in the population is *evaluated*. Each of them is assigned a quality value, resulting in a sorted population of candidate solutions. In general all four process model quality dimensions are used. But additional quality dimensions can be considered.

Next, the ETM framework evaluates if it should *terminate*. Different stop criteria can be applied here. Common examples are termination after a fixed number of generations, when the quality of the best candidate reached a certain minimum value, or when the quality of the best candidate does not significantly improve any more.

If none of the active stop criteria are satisfied, the ETM framework continues. First the best candidates in the population are copied to a collection called the *elite*. The candidates in the elite are not (to be) changed to ensure that the quality of the best candidate(s) does not decrease during the run of the evolutionary algorithm. Next, candidates are selected from the population. Common selection techniques are tournament selection and roulette-wheel selection [77] (see also Section 4.4.2). Using these selection techniques, candidates with higher quality scores are preferred for selection.

The selected candidates are then changed or *evolved* using different operations. The main *change operators* are the crossover and mutation operations. However, the option to replace a candidate with a new candidate can also be seen as a(n) (extreme implementation of a) change operation. The crossover change operation represents the biological breeding process. Crossover takes two candidates as parents and swaps parts between them to create offspring. Mutation is also inspired by biology and introduces smaller changes in candidates. Through mutation new genetic material is inserted. Without mutation, parts of the search space may be unreachable.

In the next step the elite candidates are again added to the population with the changed candidates, thus forming the population of the next generation. This population is then again evaluated and the algorithm continues this process until at least one of the termination conditions is satisfied.

4.2 Applications of the Evolutionary Framework

The ETM framework is applicable in many process mining scenarios. In this section we describe some scenarios where the ETM framework can be applied.

In later chapters we discuss and implement some of these scenarios. It should be noted that of course many other scenarios can be thought of.

4.2.1 Process Discovery

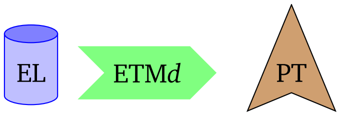


Figure 4.2: Process discovery scenario.

Process discovery is the most well-known process mining task [5]. Although many process discovery algorithms exist, many challenges remain. The ETM*d* algorithm is an implementation of the ETM framework that is able to discover a process model from an event log, while addressing some of these challenges.

The input is an event log (EL) and the ETM*d* algorithm produces a process tree (PT), see Figure 4.2. Evaluation is done by verifying how well the discovered process tree describes the behavior in the event log, using the four common quality dimensions in process discovery: replay fitness, precision, generalization and simplicity. By changing the weights of the different quality dimensions the resulting process model can be influenced. This is only possible because of the flexibility offered by the ETM framework. The ETM*d* algorithm provides a solution for Challenge 3 discussed in Section 1.3.4: balance the quality of discovered process models.

As a result of the quality evaluation, next to the discovered process tree diagnostic information is also provided. This diagnostic information describes how the behavior in the event log relates to the discovered process model. It can indicate how often certain parts are used or where the event log deviates from the process model. By providing this information the understandability of the result is increased since it indicates how good or bad the process model describes the behavior, in each of the quality dimensions. This is exactly Challenge 4 as discussed in Section 1.3.5.

The process discovery scenario is described in more detail in Chapter 5, which discusses the quality evaluation, and in Chapter 6, which implements the ETM*d* algorithm based on the ETM framework. In Chapter 7 we apply the ETM*d* algorithm on artificial and real data.

4.2.2 Process Model Repair

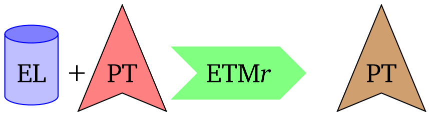


Figure 4.3: Process model repair scenario.

Process discovery is able to discover a process model from the observed behavior as recorded in the event log. However, within organizations there often already is a process model that describes how the process should be executed. Such a model is often called a reference process model, which typically describes an idealistic view of the process. As soon as this process is implemented and executed, the operational process starts deviating from this process model description.

The ETM*d* algorithm can be extended to consider this documented process model during process discovery. The resulting ETM*r* algorithm, shown in Figure 4.3, aims to discover a process model that is more similar to the reference process model. Another way to view this is that the reference model is repaired, using the observed behavior. Key here is that the resulting process model is similar enough to the original reference model.

The 'similarity' quality dimension provides control over how much the process model is allowed to be changed from the input model(s). Similarity is thus used as an additional quality dimension during the evaluation phase. The output of this scenario is a process model that is similar to the original input process model, while some changes are applied to improve the other quality dimensions used to evaluate the process model. By reducing the importance of similarity, more changes can be made to the process model, which will then better reflect the observed behavior.

In this scenario the documented process model, which contains external information, is used during process discovery. This addresses Challenge 5 (see Section 1.3.6). This scenario is described in more detail in Chapter 8 which presents the ETM*r* algorithm.

4.2.3 Process Discovery of a Configurable Model

Different organizations or units within a larger organization may need to execute similar business processes. Municipalities for instance all provide similar

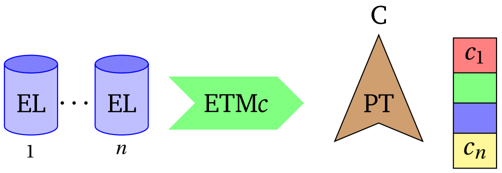


Figure 4.4: Scenario of the discovery of a configurable process model and its configurations.

services while being bound by government regulations. Within the CoSeLoG project we investigate the similarity of business processes between different municipalities (see Section 1.4). However, not only municipalities execute similar business processes. Large car rental companies like Hertz, Avis and Sixt have offices in different cities and airports all over the globe, but often there are subtle (and sometimes also striking) differences between the processes followed by these offices, even though they belong to the same car rental company. To be able to share development efforts, analyze differences, and learn best practices across organizations, we need *configurable process models* that are able to describe *families* of process variants rather than one specific process [93, 94, 154].

The ETMc algorithm is able to discover such a configurable process model by taking multiple event logs as input, as shown in Figure 4.4. Configurations are added to the basic process tree notation, describing which behavior is not allowed for a particular configuration. The ETMc algorithm then evolves a process tree, together with its configurations. Evaluation is done using the resulting process model, after the configurations for the individual input event logs have been applied. These evaluations per event log are then aggregated to an overall quality score. Also, the quality of the configurations itself is considered, for instance by counting the number of configuration points. This scenario is described in more detail in Chapter 9 which presents the ETMc algorithm in more detail.

The result is one process tree with a configuration for each of the input event logs. Additionally, diagnostic information is provided that shows how the behavior of an event log relates to the configured process model. The ETMc algorithm thus addresses Challenge 6 which is the challenge of describing a family of processes.

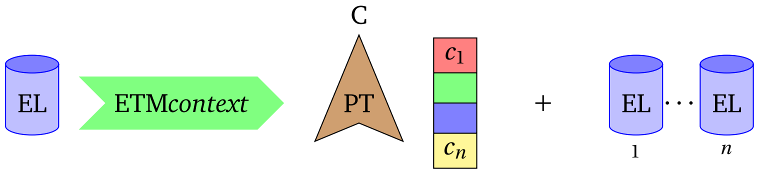


Figure 4.5: Scenario of the discovery of a configurable process model, its configurations and the context of the log splitting.

4.2.4 Configuration Discovery using Context

In the previous scenario multiple event logs were the input of the ETM framework. Each of the input event logs described a particular context of the process: a different organization or unit within an organization. However, the execution of a process might also differ because of other reasons. Different case characteristics might for instance influence the flow of a case through the process.

In this scenario the ETM*context* algorithm not only discovers configurations, but also the conditions when to apply these, as shown in Figure 4.5. These conditions are context dependent, for instance using case properties such as case type or customer history. This provides insights into the different classes of cases and how they are handled. Another example of a possible context is the employee who performs a particular activity during the execution of the case.

The result of this scenario is a process tree with diagnostic information, plus different distinguishing contextual characteristics and their corresponding process model configurations to describe the different behaviors. Therefore, this scenario can be seen as another way to approach Challenge 6 by observing that within one process many process variants are executed. This scenario is discussed and implemented in [141], but is not discussed further in this thesis.

4.2.5 Concept Drift

Concept drift is the phenomenon where the way a process is executed changes over time [40, 42]. This can be caused by the process model being altered or by a system update. Where the previous two scenarios identified the organization or case characteristics as the distinguishing factor, in this scenario time is considered as the distinguishing case factor.

The difficulty in this scenario is that there are different types of concept drift [40, 42]. *Sudden drift* indicates that one process model is changed to an-

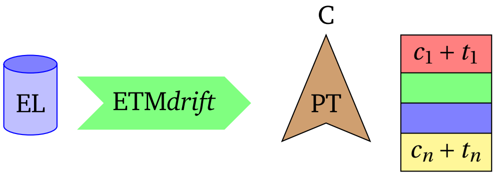


Figure 4.6: Scenario of the discovery of a configurable process model, its configurations and the time information of the log splitting.

other process model at a particular point in time. All cases, both new and currently running, now follow the new process model. *Gradual drift* is the situation where a process model is changed but only newly arriving cases follow this new process model. Cases that are already running keep following the original process model. In the case of *recurring drift* different process model variants re-occur and alternate with each other, for instance because of seasonal influences. *Incremental drift* is the concept drift type where there is a series of gradual changes from one model to another.

Sudden drift is the easiest type to detect since it involves splitting the event log on a certain point in time. All events executed after that point in time follow the new process model, which makes the relation between observed events and which process model is followed clear. With gradual drift only new cases as of a certain point in time should follow the new process model. This makes it harder to detect at which point in time the drift took place, but splitting the event log once this point in time is known is easy. Recurring and incremental drift are usually sudden and have multiple change points.

In this scenario the input is a single event log. The ETM*drift* algorithm, as shown in Figure 4.6, tries to detect points in time where the process changed, and then discovers a process model configuration for each of these time periods. Therefore, the end result is a process model with configurations, each with a point in time as of which it should be applied. This can be seen as a solution to Challenge 6 because different process versions can also be considered a family of processes. Of course, also in this scenario diagnostic information is provided that shows how the observed behavior relates to the process model and its configurations.

The difficulty with correctly implementing the ETM*drift* algorithm using the ETM framework is the detection of the different types of drift. Of course, existing techniques can be used to estimate the point in time where the process changed [40, 42]. Additionally, the change point and drift type need to be applied to the input event log.

4.2.6 Decision Mining

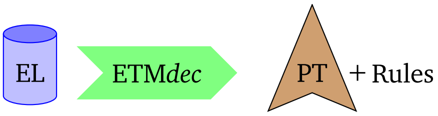


Figure 4.7: Decision mining scenario.

In this scenario the ETM*dec* algorithm, shown in Figure 4.7, not only considers the control flow but also the data flow of the process. Using a given event log, and possibly a process model, the data conditions for the choices in the process are discovered. This means that using data attributes of both the case and the current activity, the conditions for the choice to be made are discovered [156].

Existing evolutionary decision mining techniques can be applied [36]. Evolutionary decision-tree mining has several advantages over structured top-down recursive approaches. The main benefits are the ability to discover more complex conditions and prevent partitioning of the data set into too small data sets for attribute selection.

The outcome of this scenario is a process model with conditions that indicate, based on data attributes, which choice in the control flow is made for each of the points of choice in the process model.

4.2.7 Other Perspectives

In this section we mention several other scenarios that are possible, but which are not discussed in detail.

Stream Mining

Sometimes the amount of event data provided is too much to store and process off-line. In this case a full event log cannot be maintained because of the vast amounts of data. Therefore, only an event stream, with current event data, is available. This is known as stream mining in the data mining field [85], and has already been applied to process mining in [55].

Since the evolutionary framework can work using existing process models, i.e. the 'repair' scenario, applying the framework not on a static event log but on a stream of event data should be possible. Based on the incoming event stream, changes are made to the process models currently in the population. The sensitivity of the model, indicated by the amount of changes allowed within a certain time frame, can be adjusted. This allows the ETM framework to show a very recent and volatile version of the process model or the more long term behavior of the process.

In this scenario the output is a process model that changes over time as the event stream describes more or different behavior. Again, diagnostic information about the behavior of the process is recorded, although on a higher level than for event logs because of the amount of data. Also, the diagnostic information might only cover a certain time period.

One of the key characteristics of stream mining is that the stream of data is so large that it cannot be stored. A challenge in this scenario therefore is the characteristic of evolutionary algorithms that they tend to be slower than conventional algorithms. At the same time, the ETM framework is able to improve on a process model using new data, and therefore can use the currently known process model.

Enrich Process Model with Simulation Properties

In the event log a lot of information is available that can be used to create or enrich simulation models [158]. Using simulation models the effects of process changes can be simulated and forecasted. However, there is not always enough information available to make the simulation model as rich as desired. For instance, resource availability and other contextual information are typically not recorded in the event log. Filling in the missing information is not easy and assuming incorrect resource behavior can have drastic effects on the simulation results.

By using the information in the event log, the ETM framework is able to estimate these parameter settings. Then, by running 'mini simulations' these parameter settings can be compared with the actual values in the event log. This results in a process model with rich simulation properties. Of course, care should be taken when using these simulation models. Therefore, the quality of the simulation models should be thoroughly assessed not only by the ETM framework but also by the end-user.

Since the evaluation of candidates relies on running 'mini simulations', the performance of the ETM framework in this scenario heavily depends on the time required to run and analyze these mini simulations.

Social Teams Discovery

Besides control-flow and data-flow aspects, the social aspects of processes [19, 150] can also be considered by the ETM framework. For instance by evaluating how social teams can be formed between resources. Using the information stored in the event log, relationships between (groups of) resources can be discovered. By using the discovered process model, more information about the social aspect of the process can be derived.

Additional quality metrics can be used for evaluating how the teams of resources should be formed. Examples of such metrics are (to minimize) the number of handovers between teams, and set requirements on the team sizes [150]. This results in a suggestion on how to group the resources used in the process in teams, based on these different metrics.

4.2.8 Combinations of Scenarios

All the scenarios mentioned in this section can be combined to form new scenarios. For instance the 'Process Model Repair' and 'Process Discovery of a Configurable Process Model' scenarios can be combined. The resulting scenario is about repairing an existing configurable process model. Another variant of the same combination of scenarios is the discovery of a configurable process model where the configured process models are similar to the individual process models currently known within the organization.

Many other combinations and extensions of the scenarios mentioned in this section can be made. This illustrates that the proposed framework is very general. Later chapters will provide concrete examples for selected scenarios.

4.3 General Requirements for Evolutionary Algorithms

Several general and key requirements should be adhered to for evolutionary algorithms to perform well. Each step of the evolutionary algorithm should consider these requirements in order for the whole algorithm to work. In this section we address some of the most important requirements for, and threats to, the ETM framework. There are however many more aspects to consider that are general to any type of evolutionary algorithm [35, 77].

4.3.1 Population Diversity

From a high-level perspective an evolutionary algorithm should balance two key aspects of search: *exploration* and *exploitation* [77]. The exploration aspect is visiting new untested regions of the search space. Exploitation is when the search concentrates on the vicinity of known good solutions to further optimize candidates. There is a clear trade-off between these two aspects. If too much time is spent in exploration the search is inefficient. On the other hand, if too much time is spent in the exploitation phase, the search might get stuck in a suboptimal set of process models. This is called *premature convergence* [77] where population diversity is lost too quickly which causes the search to get trapped in a *local optimum*.

Evolutionary algorithms also suffer from what is known as *anytime behavior* [77]. The overall quality of the best candidate often rapidly increases in early generations. However, after some time the best quality score only marginally improves. This means that the search can be stopped at any time and the algorithm will have a solution, albeit suboptimal. It also means that running an evolutionary algorithm for very long is often not worth the amount of time and effort spent considering the marginal increase in quality. Furthermore, it indicates that spending much effort in optimizing the initial population is often not worth the effort since the evolutionary algorithm very quickly finds good candidates.

To combat both premature convergence and anytime behavior, it is important that the population is both diverse and not too diverse. In order to correctly balance the trade-off between exploration and exploitation, the population should be very diverse in the initial generations. To prevent premature convergence the population should stay sufficiently diverse after the initial generations. At the same time, the evolutionary algorithm should be able to perform exploitation which requires more similar candidates to exist in the population to allow for small optimizations (i.e., local search).

4.3.2 Ability to Visit the Whole Search Space

As with all search algorithms, an evolutionary algorithm should be able to visit the whole search space as described by the representation chosen. Having a diverse population alone is not enough. The evolutionary algorithm should also be able to cover the whole search space, given enough time [35, 77]. For initial candidate creation this means that the whole search space should in theory be covered by running the initial creation algorithms indefinitely. Crossover and mutation together should cover the whole search space starting from any initial population. Crossover usually causes bigger jumps in the search space than mutation since it makes bigger changes to the model. Mutation should allow for a more fine-grained change of the candidate. This means that enough randomness should be introduced when creating and changing candidates. An evolutionary algorithm without randomness, so with only smart or guided creation and change operators, is unable to discover novel solutions to the problem. Therefore *randomness* is an important factor in evolutionary algorithms.

4.3.3 Prevention of Bloat

A common problem in genetic programming algorithms is that the size of the trees in the population grows over time (this is also known as "Survival of the fattest") [35, 77]. Bloat is mainly caused by the introduction of introns [35], which are pieces present in the candidate that have no effect on the quality of an individual. Although introns do not directly contribute to the quality of an individual, introns do increase the likelihood that descendants of the individual have a better quality. However, introns do not contribute to the direct quality of an individual and therefore should not be present any more in the final results of the evolutionary algorithm. Different methods exist to prevent bloat, one of the easiest methods is preferring smaller solutions over bigger ones in the evaluation criteria.

4.3.4 Requirements for the Evaluation of Candidates

Evaluation of candidates is an important aspect of any evolutionary algorithm. Several metrics can be created to indicate the quality of a candidate. Since there are multiple quality dimensions in process discovery, multiple metrics are combined in the ETM framework. However, there are some important requirements that each metric should follow.

Efficient Implementation For analysis purposes the quality of a process model in relation to an event log is in general only evaluated once and on request. For some metrics the easiest way of calculation often is also the most time-consuming one. The answer should be provided in a timely manner when an analysis is requested. However, evolutionary algorithms, such as the ETM framework, evaluate many different candidate solutions. Therefore, the performance of an evolutionary algorithm mainly depends on the time required by the metrics to evaluate the candidates [105, 110]. Approximation of the quality may greatly improve the performance of algorithms depending on the metric. Of course the quality of the approximation should be good enough and not introduce an undesirable bias towards particular suboptimal solutions.

Intuitive Results Another important requirement for metrics is that the results are intuitive [105]. This can be achieved by making the quality metrics continuous: small improvements should result in small value changes and big improvements in big value changes [35]. Why the quality according to the metric is better or worse, and by 'how much', should also follow the philosophy of the quality dimension [35, 155].

Additionally, the metric should be *repeatable* and return the same result every time the metric is calculated on the same input [155]. If results are different between different calculations on the same input, the results are not reliable enough.

Finally, the evaluation criteria should *not contain loopholes*. Since evolutionary algorithms are excellent at optimizing candidate solutions for the given evaluation criteria, chances are that the algorithm finds and abuses special cases that are not considered by the metric. Although the quality dimension seems improved by abusing these loopholes, intuitively the quality dimension did not improve. Therefore it is important that quality metrics contain no loopholes that can be abused by the evolutionary algorithm.

Clear Specification The specification of the metric, i.e., the way it is calculated, should also be clear. If the specification cannot be understood, verifying why a certain process model has a certain value assigned to it can not be done. Furthermore, the metrics should require as few parameters as possible since unknown parameter values make interpreting the results difficult. Moreover, in certain situations, parameters can change the resulting value of a metric dramatically. This makes a metric unclear and less authoritative. The metric should be robust to different situations without requiring parameters.

Orthogonal Different metrics should be orthogonal to each other, in order to be used in combination. If two metrics both punish or reward the same aspect of a process model then that aspect is likely to be over-emphasized. This should be expressed by aiming for a good score for a single metric that considers this aspect. If two metrics overlap, the results become unclear and one of the metrics is redundant. Furthermore, each metric should only cover a single quality dimension for the same reason.

An exception to this rule is incorporating additional specific preferences as stated by the end-user. Examples are preventing the use of loop constructs or controlling the repetition of activities. Those preferences are rarely independent of functional requirements of the process model. They should however be incorporated if the end-user desires it.

4.4 Common Implementations of the Phases of an Evolutionary Algorithm

In this section we discuss several common implementations for the different phases of the ETM framework: *candidate evaluation*, *selection*, *change* and *termination*.

4.4.1 Candidate Evaluation

Evaluation of candidates is an important aspect of evolutionary algorithms since it defines what improvement means. The evaluation function determines what the result of an evolutionary algorithm is. Since the evaluation heavily depends on the internal representation used and on the problem that is to be solved by the evolutionary algorithm, no standard evaluation implementation exist.

A common phenomenon in candidate evaluation however is the combination of several evaluation functions to consider different aspects of the candidates [60]. This is commonly known as multi-objective optimization. Since a single evaluation value has to be provided for the evolutionary algorithm, these different values have to be combined into a single value. The most used approach to do this is by taking the *weighted average* of the different values. This provides the freedom to assign different weights to different evaluation functions. However, this method has several drawbacks:

1. Determining the correct weights for the different quality dimensions upfront is difficult. Several characteristics of the event log have an effect on the value for the different quality dimensions. It is however impossible to estimate beforehand what those effects are.

2. Values need to be normalized for comparison: for the weighted average the values are usually normalized. However, dimensions can still respond differently to changes. Furthermore, a normalized value often provides less information than an absolute value and interpretation may be difficult.

3. Only one solution is provided: only the candidate with the best weighted average is presented. However, no insights into the different trade-offs among the dimensions are provided.

The so-called *Pareto front* is often used as an alternative to the weighted average [43, 60, 171] since it is a model that supports reasoning about multi-objective optimization trade-offs. The general idea of a Pareto front is that all members are *mutually non-dominating*. A member dominates another member if for all quality dimensions it is at least equal to or better, and for one dimension strictly better, than the dominated member. Since all members in the Pareto front are mutually non-dominating (none of them dominates another member) they represent different trade-offs in the quality dimensions, sacrificing one quality dimension to improve another. This concept was originally proposed by Vilfredo Pareto to explain economic trade-offs [146].

An example of a Pareto front in two dimensions is shown in Figure 4.8. Each dot in the graph represents a process model with a certain replay fitness and precision value. For each dimension a bigger value indicates a better candidate, i.e., the goal is to obtain a process model in the top right corner of the chart. However, often there is no single model that is able to score perfectly on all quality dimensions. The unfilled dots in the lower middle area of Figure 4.8 are non-optimal process models, i.e., one of the dimensions can be improved without reducing the quality in (any of) the other dimension(s). The closed black dots represent the current estimation of the Pareto front. For these process models there is currently no model known where one dimension has a better score without a reduction of the quality in the other dimension. The bigger dots show the nine most diverse process models in the current front, which can be used to truncate the Pareto front by keeping only one representative for a group of similar process models. The ideal or real Pareto front, as indicated by the curved line, shows that some improvements can still be made by continuing the evolutionary search. Of course in practice the ideal Pareto front is unknown.

The ETM framework as shown in Figure 4.1 can be extended with a Pareto front cache that maintains the current Pareto front during the different generations of the ETM framework. At the end of each generation the current evolved and evaluated population is added to the Pareto front. All candidates in the

Pareto front that are dominated by other candidates in the front are then removed from the Pareto front. At the beginning of the next iteration a fixed number of candidates is selected from the Pareto front, since the front can grow larger than the desired population size. When the ETM framework terminates, the whole Pareto front of process models is returned, instead of only a single process model. This way the user can be involved in the final process model selection.

4.4.2 Selection

During the selection phase candidates are selected from the current population for further evolution. However, to prevent a reduction in the quality of the best candidate, a group of *elite* candidates is created first. Usually a fixed number of best candidates is cloned from the current population into the elite group of candidates. After the change phase, these elite candidates are added back to the population.

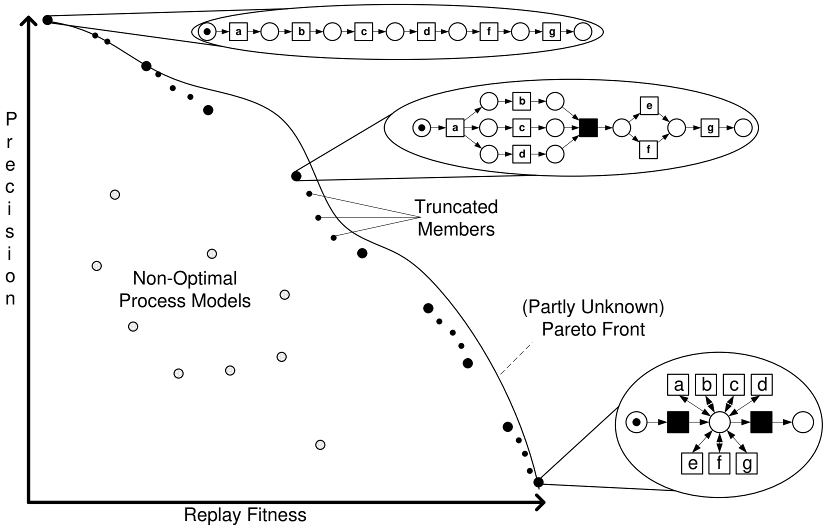


Figure 4.8: Pareto Front of the two quality dimensions replay fitness and precision. The hollow dots are non-optimal process models, the small black dots are discovered process models representing the current Pareto front and the big black dots are the 9 most diverse among the currently known process models.

The remaining part of the new population is created by changing candidates that are selected from the current population. These selected candidates are changed in the next phase of the evolutionary algorithm, to improve the quality of the candidates. Determining which candidates are selected for further evolution should be a balance between the exploration and exploitation aspects of search (see Section 4.3.1): if population diversity is lost too quickly the algorithm might be trapped in a local optimum. However, if too much diversity is maintained, the algorithm is not able to perform small optimizations to fine-tune the candidates.

Several selection strategies exist, most of which consider the quality assigned to a candidate to maintain population diversity. The most common and well-known selection strategies are [77, 100]:

Fitness Proportional Selection randomly selects a candidate from the population with a probability proportional to the quality value of the candidate as compared to the quality values of the rest of the population.

Sigma Scaling Selection is an extension of the fitness proportional selection strategy and uses information about the mean and standard deviation of the quality of the whole population to be more likely to select candidates with a relatively high fitness.

Ranking Selection first sorts candidates on quality, and then randomly selects a candidate where a candidate with a high rank is more likely to be selected. In this strategy the absolute difference between the quality values has no influence on the likelihood of selection.

Roulette Wheel Selection is a method that tries to select candidates in such a way that the selected sample has the same characteristics as the whole population. This is achieved by assigning candidates a chance of selection proportional to the quality of the candidate. A way to visualize this is by assigning candidates slices of a roulette wheel proportional to their quality. Roulette wheel selection then spins the roulette wheel as many times as the number of candidates that need to be selected.

Stochastic Universal Sampling assigns each candidate a chance of selection proportional to their quality, like roulette wheel selection does. However, stochastic universal sampling randomly chooses a point on the roulette wheel to start from, and then advances equally sized steps around the roulette wheel to select the number of candidates required. This ensures equal spread of the candidates selected without any bias [34].

Tournament Selection does not use information about the entire population, as the previous strategies do. Instead, tournament selection randomly selects a specified number of candidates and then only returns the one with the best quality. This process is repeated until the desired number of candidates is selected.

In case a Pareto front is discovered an overall quality value needs to be assigned to a candidate that expresses the quality of that candidate considering the Pareto front. This is necessary to select the best candidates from the Pareto front which serve as input of the new generation. Different approaches exist [61, 188] to assign such an overall quality value to candidates in a Pareto front. The general idea of all approaches is the same: unique candidates should have a higher chance of selection. Looking at the example Pareto front as shown in Figure 4.8, the process models at the extremes of both quality dimensions have a high chance of being selected. Also the thicker black dots have a higher chance since they are on the edge of a group of similar process models. This makes sure that the population that is selected from the Pareto front is diverse, which increases the chance of discovering a new good candidate. The quality value calculated for each candidate in the Pareto front is then used as input for the selection strategies discussed.

4.4.3 Change Operations

Evaluating and selecting candidates alone is not sufficient for evolutionary processes. The candidates also need to be created and changed, so that new candidates can be found and evaluated. Three types of change operators exist: candidate replacement by new candidates, candidate crossover and candidate mutation.

Together, the change operations of an evolutionary algorithm need to ensure that the whole search space can be covered. This means that by combining several executions of the different change operations, every candidate should be possible to create.

Candidate Creation and Replacement

One of the change operations that can be applied is selecting certain (particularly bad) candidates to be replaced by newly created candidates. Candidates can be created using the same techniques that are used at the start of the evolutionary algorithm to form an initial population of candidates.

Candidates can be created randomly or by using heuristics to aim at a higher quality. The key aspect here is that the created candidate does not need to have a good, or even reasonable, quality, since the quality will be improved during coming generations. Adding variability to the population is more important than creating initially good quality candidates.

Crossover

A commonly used operator in evolutionary algorithms is the crossover operation. This operator is inspired by the mating process of species. Generally speaking a crossover operator takes two candidate solutions and exchanges information from the two parents into the two offspring candidates. The choice of which information to exchange, and the way the information is inserted into the offspring candidates, varies between implementations.

The main idea behind crossover is that good but different parts of two candidates are combined into offspring. This approach works well in the real world for breeders of plants and livestock to produce species that have higher yields or other desirable features [77]. Within evolutionary algorithms the offspring created by random combination might not always result in offspring that has better quality than their parents. However, if the offspring is better than its parents, it often is significantly better. Unlike nature however, crossover is generally applied probabilistically.

Mutation

The mutation change operation takes a single candidate solution and creates a (slightly) modified child of it. The most common types of change applied are the addition, removal and modification of parts of the candidate solution. Although this change is usually applied randomly, some mutation operators might incorporate heuristics to mutate the candidate in a more targeted fashion. Important for this operator type however is that there should be enough randomness involved, in order to keep diversity of the population sufficiently high.

4.4.4 Termination

Unlike evolution in nature, an evolutionary algorithm has to terminate at some point. However, since an optimal solution is unlikely to be found, termination criteria need to be used. The most commonly used termination conditions are:

Quality Threshold terminates when the best, or average, quality has reached a certain threshold.

Number of Generation Threshold terminates when a predetermined number of generations has been executed.

Elapsed Time terminates when a predetermined amount of time has passed.

Quality Stagnation terminates when the best or average quality does not change significantly for a number of generations.

User Cancelation terminates the evolutionary algorithm when the user manually cancels the execution.

An interesting feature of an evolutionary algorithm is that it can always return the best solution found so far. This means that intermediate results can be inspected and used.

4.5 Conclusion

In order to balance the different quality dimensions during process discovery, a flexible process discovery algorithm is required. Therefore this chapter presents the evolutionary process mining framework called the ‘Evolutionary Tree Miner’, or ETM for short. The general phases of the ETM framework demonstrate the flexibility of the framework. The applicability of the ETM framework is shown by several scenarios discussed in Section 4.2.

Next, we discussed important requirements for an evolutionary algorithm. Population diversity is an important requirement that should be balanced to allow for the discovery of the optimal solution. Furthermore, the evolutionary algorithm should be able to visit the whole search space to be able to find the optimal solution. We also discussed several requirements for the evaluation of candidates.

Finally, we discussed common implementations of and approaches to the different phases of the ETM framework. One of these approaches was the construction of a Pareto front to keep the individual quality metrics separate. Moreover, it allows for multiple candidate solutions to be returned, each with different trade-offs.

In the next chapter we discuss several quality dimensions and quality metrics that can be used for process discovery. This is followed by Chapter 6 where we present the ETM*d* algorithm which is the primary implementation of the ETM framework.

Chapter 5

Process Model Quality Dimensions

As discussed in Challenge 3, the four different process model quality dimensions should be considered during process discovery. Furthermore, for discovered process models, how well they describe the event log should be indicated using the same quality dimensions. In the previous chapter we presented a flexible evolutionary process discovery framework that is able to incorporate these quality dimensions during discovery. The way process model candidates are evaluated in the evolutionary algorithm determines which process models survive and are returned by the algorithm. Therefore, the four quality dimensions need to be well understood before they can be measured.

In Section 5.1 we first discuss the four well-known quality dimensions for process discovery. Section 5.2 provides a more theoretical view on the quality of a process model, taking not only the event log, but also an omnipresent but unknown system into consideration. This is followed by a discussion of several ways to measure each of the four quality metrics in Section 5.3 through Section 5.6. Section 5.7 discusses why all four quality dimensions are necessary. Section 5.8 discusses some of the considerations taken into account in the quality metric selection. Section 5.9 discusses the possibility to add quality dimensions. Section 5.10 summarizes related work and Section 5.11 concludes this chapter.

5.1 The Four Process Discovery Quality Dimensions

Given an event log, several process models can be presented that describe the behavior as recorded in that event log. Figure 5.1 shows the four quality dimensions that are typically considered when evaluating these process discovery results [5, 8, 10]. The dimension of *replay fitness* quantifies the fraction of the event log supported by the process model. *Precision* quantifies how much of the behavior described by the process model is not observed in the event log. The dimension of *generalization* quantifies the likelihood of previously unseen but allowed behavior being supported by the process model. Finally, *simplicity* evaluates the complexity of the process model, where simpler models are preferred over more complex ones, as per Occam's Razor [5].

The quality dimension of simplicity is evaluated without explicit use of the event log. The three quality dimensions of replay fitness, precision and generalization use the behavior recorded in the event log to evaluate the process model. In this chapter we first reconsider the notion of quality of a process mining result, by explicitly assuming the notion of a "system" outside the process model. In previous work the system was not explicitly considered when determining the quality of a process mining result [5, 10, 21, 27, 58, 89, 155, 157]. Nevertheless, the notion of a system is often implicitly assumed when discussing quality issues. A system can be a concrete information system implementation but usually refers to the context of the process, e.g., the organization, rules, economy, etc. This system may allow people involved in the operational process to deviate from the intended behavior of the information system, sometimes for

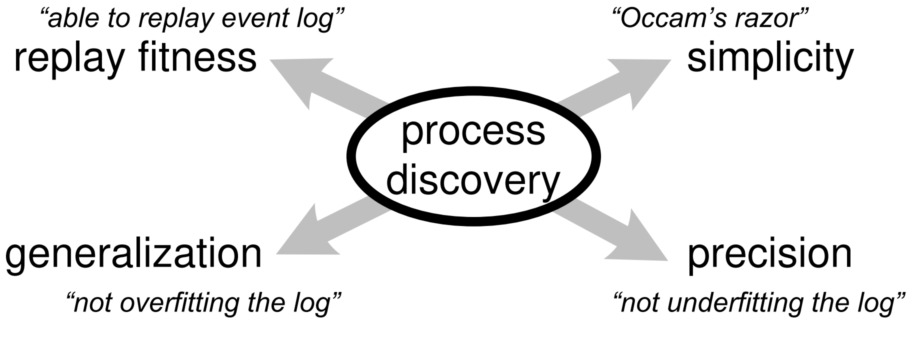


Figure 5.1: The four quality dimensions for process models in process discovery (from [5]).

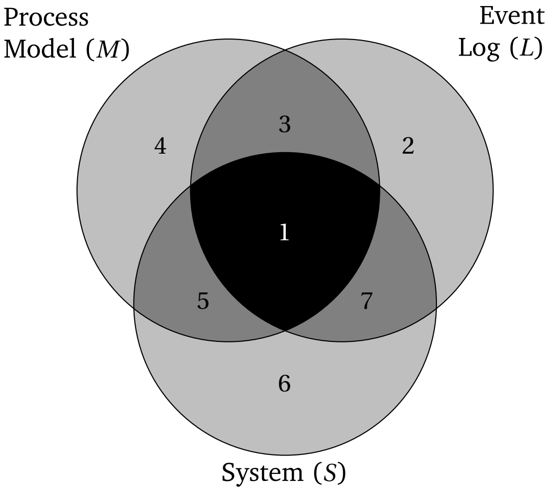


Figure 5.2: Venn diagram showing that the behavior of the process model (M), event log (L) and system (S) can be disjoint or overlapping.

good reasons. We show that explicitly considering the presence of such a system leads to new insights into the role of existing quality dimensions in process mining.

5.2 Theoretical View

In Figure 5.2, we explicitly depict the behavior of a process model, the behavior observed in the event log and the behavior allowed by an observed system. As shown, the behavior included in these three entities can partially overlap. In practice there are many forms in which behavior can be described, such as traces [8] (see Section 2.3 and Section 3.2), behavioral profiles [179], α-relations [20] or one of the many process modeling notations (see Section 2.2). In the case of process trees for instance the language of the root of the process tree (i.e., $\mathscr{L}(r)$) can be used. However, for the discussion that follows, we abstract from the way the behavior is exactly described. The only assumption we make is that the 'amount of behavior allowed' can be counted. Especially in cases of loops, which theoretically allow for infinite behavior, an estimation on the size of the behavior is assumed to be available.

5.2.1 Relating the Behavior of the Event Log, the Process Model and the System

The Venn diagram shown in Figure 5.2 shows seven areas. We can intuitively describe the behavior contained in each area as follows:

1. **Modeled and observed system behavior ($L \cap M \cap S$).** The central black area in the Venn diagram contains all behavior of the system that is also observed in the event log and is possible according to the process model.

2. **Not modeled but observed exceptions ($(L \setminus M) \setminus S$).** All the observed behavior that is actually non-system behavior is considered an exception. The exceptions that are not supported by the process model are contained in this area.

3. **Modeled and observed exceptions ($(L \cap M) \setminus S$).** Modeled and observed exceptions are those exceptions observed in the event log that are described by the process model.

4. **Modeled but unobserved and non-system behavior ($(M \setminus S) \setminus L$).** This contains all the behavior described by the process model which is non-system behavior and is also not found in the event log.

5. **Modeled but unobserved system behavior ($(M \cap S) \setminus L$).** The behavior described by the process model that is the system's behavior but is not seen in the event log.

6. **Not modeled and unobserved system behavior ($(S \setminus L) \setminus M$).** All the system behavior that is neither observed in the event log nor modeled by the process model.

7. **Not modeled but observed system behavior ($(S \cap L) \setminus M$).** The system behavior that is observed in the event log but not described by the process model.

It is important to realize that there is generally no way to explicitly describe the behavior of the system, first of all since this behavior is typically infinite (e.g. due to loops), but more so because there is always the possibility of unforeseen behavior in any real-world system. In fact, systems tend to change over time. Nonetheless, the traditional goal of process mining is to *find a process model that describes the system as accurately as possible, using nothing more than the observed behavior in the log*. In the remainder of this section, we assume that the behavior of the system is known. In Section 5.2.2 we discuss how to deal with the fact that the behavior of the system is generally not known.

By relating the behavior allowed by the process model to that recorded in the event log, we can distinguish two metrics commonly used in information retrieval. The *precision* between the process model and the event log expresses the amount of behavior that can be produced by the process model but is not seen in the event log. This can be expressed as[1]:

$$\text{Model-log precision} = \frac{|L \cap M|}{|M|} \tag{5.1}$$

The *recall* between the model and the event log quantifies the behavior of the event log that can be produced by the process model compared to all the observed behavior in the event log:

$$\text{Model-log recall} = \frac{|L \cap M|}{|L|} \tag{5.2}$$

Precision and recall between the process model and the event log follow the common notions of precision and recall in information retrieval. However, in process mining the problem setting is a bit different. In information retrieval, instances should be classified correctly in a large set of instances. In process mining however, we have instances of observed behavior, as recorded in the event log, that the process model should describe. This event log is created by, or obtained from, a system. So the behavior observed in the event log can also be related to the (actual) behavior of the system. This can be expressed by precision between the event log and the system:

$$\text{Log-system precision} = \frac{|L \cap S|}{|L|} \tag{5.3}$$

This expresses the fraction of the observed behavior in the event log that is included in the system.

The amount of overlap between the observed behavior recorded in the event log and the behavior of the system can be expressed as follows:

$$\text{Log-system recall} = \frac{|L \cap S|}{|S|} \tag{5.4}$$

[1] Note that we assumed that the ‘amount of behavior’ can be counted.

This expresses the fraction of the behavior of the system that is seen in the event log, with respect to all the behavior of the system.

The behavior allowed by the process model can also be compared to the behavior of the system. Again, precision can be calculated, but now for the process model with respect to the system. This is expressed as:

$$\text{Model-system precision} = \frac{|S \cap M|}{|M|} \tag{5.5}$$

This fraction thus expresses the fraction of behavior that is allowed by the process model but is not part of the behavior of the system.

Finally, recall between the model and the system expresses the fraction of the behavior expressed by the process model that is also the behavior of the system:

$$\text{Model-system recall} = \frac{|S \cap M|}{|S|} \tag{5.6}$$

If all these six fractions are equal to one then the three circles of the Venn diagram of Figure 5.2 coincide. This means that the event log exactly captured the behavior allowed by the system, and the process model exactly describes the event log and thus the system. If all fractions are zero then the three circles are disjoint. This means that the event log contains only behavior that is not allowed by the system and the process model describes behavior that is neither present in the event log nor allowed by the system.

In process mining, we start from a given event log L which comes from a given system S, i.e., L and S are constant. If we assume that S is known then we could simply use a genetic algorithm to discover a process model M which maximizes all of the fractions. However, the behavior of the system is unknown, but can, to some extent, be estimated from L.

5.2.2 Dealing with an Unknown System

When considering the notion of a system, we rephrase the goal of process mining to: *discover a process model M from a given log L taken from an unknown but constant system S, such that M maximizes all fractions listed in Section 5.2.*

Table 5.1 shows a summary of the different precision and recall metrics that can be calculated on the Venn diagram of Figure 5.2. The table also relates the four quality dimensions shown in Figure 5.1 to these metrics.

Table 5.1: Overview of the different fractions and quality dimensions.

	Equation		Description
	-		Simplicity (see Section 5.3)
5.1	Model-log precision	$= \frac{\lvert L \cap M \rvert}{\lvert M \rvert}$	Precision (see Section 5.5).
5.2	Model-log recall	$= \frac{\lvert L \cap M \rvert}{\lvert L \rvert}$	Replay Fitness (see Section 5.4).
5.3	Log-system precision	$= \frac{\lvert L \cap S \rvert}{\lvert L \rvert}$	Observed non-exceptional behavior. This fraction is not considered here because it does not depend on the process model.
5.4	Log-system recall	$= \frac{\lvert L \cap S \rvert}{\lvert S \rvert}$	Log completeness. This fraction is not considered here because it does not depend on the process model.
5.5	Model-system precision	$= \frac{\lvert S \cap M \rvert}{\lvert M \rvert}$	Modeled system behavior. Under the assumption that replay fitness, precision and generalization are considered, this fraction is irrelevant.
5.6	Model-system recall	$= \frac{\lvert S \cap M \rvert}{\lvert S \rvert}$	Generalization (see Section 5.6).

The notion of model-log precision directly relates to the quality dimension of precision and model-log recall relates to the quality dimension of replay fitness. Furthermore, if L and S are fixed, then the log-system precision and recall are constant, i.e., these fractions become irrelevant for process discovery. In fact, most work on process mining [5,27,96,180] uses the notion of *noise* to describe exceptional behavior, i.e., behavior observed in the log, but that is not part of the system (the noise level corresponds to 1−log-system precision). Furthermore, a certain level of completeness is often assumed which refers to the completeness of the log with respect to the system, i.e., log-system recall.

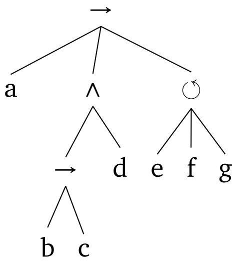
Things become more complex when we consider model-system precision and model-system recall under the assumption that the system is unknown. Basically, these metrics cannot be computed or estimated without further assumptions. Typically, process discovery algorithms use a hidden assumption that the process model they discovered from the event log does not include behavior outside of the system, i.e., they assume that $M \subseteq S$, hence model-system precision is one. This leaves the model-system recall to be estimated.

Finally, model-system recall represents the fraction of the system which is covered by the model, i.e., if this recall is high, any behavior of the system can be explained with the model, regardless of whether this is observed or unobserved behavior. Under the assumption that $M \subseteq S$, this is what is traditionally referred to as *generalization* [5, 10].

Based on the discussion in this section, in the remainder of this chapter we discuss the four well-known process discovery quality dimensions of replay fitness, precision, generalization and simplicity in more detail. However, we can now relate the three quality dimensions of replay fitness, precision and generalization to the more theoretical view discussed in this section.

Throughout the remainder of this chapter we use the running example of Figure 5.3 to further explain the meaning of these quality dimensions. We use the event log as shown in Figure 5.3b to illustrate the quality dimensions. For completeness, the process model that generated this event log is shown in Figure 5.3a. Please note that the process model in this case represents the (usually unknown) system.

In the remainder of this chapter L is represented by an event log, and M by a process model as the description of the behavior.



(a) (Unknown) system that generated the event log, represented by a process tree.

Event Log	#
a b c d e g	80
a b d c e f e g	5
a d b c e g	15

(b) Generated event log.

Figure 5.3: Running example used to explain the quality dimensions in process discovery.

5.3 Simplicity

The quality dimension of simplicity quantifies the simplicity of the process model and therefore is the only quality dimension that is not necessarily related to the behavior of the process model or event log. Simplicity of the process model is defined by two aspects. The first aspect is related to *Occam‘s Razor* which states that “one should not increase, beyond what is necessary, the number of entities required to explain anything”. Since the other three quality dimension already evaluate “what is necessary” in the process model, simplicity mainly focusses on reducing the size of the process model. This aspect therefore also helps to prevent bloat (see Section 4.3.3).

The second aspect that can be considered in the simplicity dimension is the simplicity of the process model as perceived by a user. However, this aspect is hard to capture and measure, since this is related to the understandability of the process model. Many factors influence the understandability of a process model [152], one of which is the visualization of the process model [119]. The main reason process models are perceived to be complex however is the size of the process model [133]. Other factors are the layout of the process model and the use of certain control flow constructs. As discussed in Section 3.5, the process tree notation used in this thesis can be visualized using many process model notations. Furthermore, since process trees are block-structured, the graph-based process model translations are also structured and hence have an increased understandability. However, during discovery of a process model the representation chosen to present it to the user is unknown. Therefore this can not be incorporated in the simplicity metric. One could however restrict the use of certain constructs, such as $\vee$ or $\circlearrowleft$-operators, to improve the simplicity, or understandability, of the discovered process model.

Unlike the other quality dimensions, multiple simplicity metrics can be combined to express the simplicity of a process model.

5.3.1 Simplicity by Ratio of Useless Nodes

Although often ignored, one of the easiest ways to reduce the size of a process model is to remove elements that do not add or limit behavior. These useless nodes can be removed without changing the behavior of the process model, while reducing its size. A simple example is a τ node in a sequence, which can be removed without changing the behavior of the process model. Additionally, in the case of evolutionary algorithms, this also helps with the prevention of bloat (ref. Section 4.3.3). The reasoning behind this metric is that an activity is added to the process model only if this is beneficial for the other quality dimensions.

We therefore define simplicity as follows:

$$Q_s = 1 - \frac{\#\text{useless nodes}}{\#\text{nodes}}, \tag{5.7}$$

where a node in a process tree is useless if at least one of the following conditions hold:

1. The node is a τ node in a sequence or parallel construct;

2. The node is an operator node with only one child;

3. The node is an operator node that has only useless nodes as children;

4. The node is a τ node and is not the first τ node in an exclusive or non-exclusive choice;

5. The node is a loop consisting of only one other loop function and two τ children;

6. The node is a τ node of a parent for which condition 5 holds;

7. The node is of the same type as its parent (unless the node is an $\circlearrowleft$-operator or if the process tree is configurable (see Chapter 9)).

Figure 5.4 shows a process tree with examples of useless nodes, as indicated by the gray circles. This process tree contains 6 useless nodes, out of 19 nodes in total, hence the simplicity is:

$$Q_s = 1 - \frac{6}{19} = 1 - 0.3158 = 0.6842. \tag{5.8}$$

Useless nodes can help candidate solutions to improve in later generations. However, the goal is to prevent or reduce the number of useless nodes in the end result.

The useless nodes metric obeys all metric requirements as stated in Section 4.3.4. Deciding whether a node is useless or not is a simple evaluation, which can be performed quickly.

5.3.2 Other Simplicity Metrics

Different simplicity metrics exist that take the size of the process model into consideration. Even more simplicity metrics can be derived from the observations in [133].

Simplicity by size

One of the simplest ways of quantifying the size of a process model is by just counting the number of nodes in the process tree. This can be normalized to return a value between zero and one, by dividing 1 by the size of the process model. However, one could argue that the size of a process model should be relative to the number of activities in the process. The benefit of this simple simplicity metric is that it adheres to all quality metric requirements.

Simplicity by activity occurrence

Another way to evaluate the size of the process tree is to consider the activities present in the event log. If each activity is represented exactly once in the process tree, that process tree is considered to be as simple as possible. Therefore, simplicity is calculated as follows:

$$Q_s = 1 - \frac{\#\text{duplicate activities} + \#\text{missing activities}}{\#\text{nodes in process tree} + \#\text{event classes in event log}} \tag{5.9}$$

Duplication of activities is measured by counting the number of times the activity is repeated in the process model. An activity is missing from the process

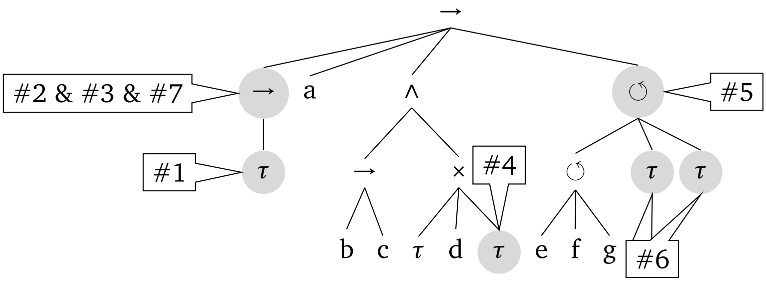


Figure 5.4: Process tree with several useless nodes (marked with gray) and the reason why they are useless (indicated by the callouts).

model if it is not included in the process model while it is present in the event log. These numbers are summed up and normalized by the total number of nodes in the process tree and event classes (or activities) in the event log.

This simplicity metric adheres to all requirements except the orthogonality requirement. By forcing each activity to be in the process model exactly once, the other three quality dimensions are influenced. On some occasions it could be beneficial for the other quality metrics to either exclude or duplicate an activity to increase their quality dimension.

5.4 Replay Fitness

Replay fitness quantifies the extent to which the behavior of the event log can be replayed in the process model. Replay fitness therefore evaluates the following fraction:

$$\text{Model-log recall} = \frac{|L \cap M|}{|L|} \tag{5.2 repeated}$$

The meaning of this quality dimension can be best explained with a simple example, as shown in Figure 5.5. The process model only allows for the sequential execution of activities a, b, c, d, e and g. Although this is exactly the first trace of the event log of Figure 5.3b, the other two traces do not fit this model. Several techniques exist to detect and evaluate these mismatches between the event log and the process model.

5.4.1 Alignment-based Replay Fitness

The main challenge for replay fitness metrics is how to relate the observed events in traces to nodes in the process model. Especially in situations where the process model and the trace are in disagreement, it is crucial how this is evaluated by the replay fitness metric. The most robust, flexible, and state-of-the-art

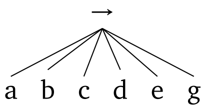


Figure 5.5: Process model without perfect replay fitness.

metric for replay fitness is based on searching for these optimal alignments between traces of an event log and a process model [10,21,22,24]. Basically, this technique aligns as many events as possible from the trace with activities in an execution of the model (this results in a so-called *alignment*).

The goal is to find the optimal alignment that minimizes the total cost. Each time the trace and the process model are 'out of sync', i.e., only a move on either the log or model is made, the cost of the alignment increases. This approach allows us to define different costs per activity and move type. By default deviations from the trace have higher cost than deviations from the process model since in general the traces are trusted more than the process model. The default cost when only a move on the model is made is 2 and when only a move on the trace is performed the cost is 5.

The alignments for the traces of Figure 5.3 on the example process model of Figure 5.5 are shown in Table 5.2. An alignment aligns the events of the trace (the top row in each alignment in Table 5.2) with the (enabled) activities in the process model (the bottom row in the alignment). The alignment between the trace $\langle a, b, c, d, e, g \rangle$ and the process model is shown in Table 5.2a. The alignment is perfect in the sense that each event in the trace can be matched with an enabled activity in the process model. Moreover, when the trace is finished, so is the process model. When an event in the trace can be matched with the execution of an enabled activity in the process model this is called a *synchronous move*.

The trace $\langle a, d, b, c, e, g \rangle$ however cannot be perfectly replayed in the process model. The alignment of Table 5.2b shows that event `d` cannot be replayed correctly since it occurs too early in the trace. This is indicated in the alignments by moving forward on the trace with `d`, while the process model does not move, as is indicated by the ≫-symbol. When this happens this is called a *move on log only*. However, later when the process model can and must execute `d`, the trace cannot. In this case in the alignment the trace does not make a move as is indicated by the ≫-symbol for the trace. Since only the process model makes a move, this is called a *move on model only*. The alignment contains one move on log only and one move on model only, and therefore the cost of the alignment is $5 + 2 = 7$.

The alignment for the trace $\langle a, b, d, c, e, f, e, g \rangle$ on the process model is shown in Table 5.2c. In total four deviations are recorded, where the first two are deviations of activity `d` in a similar way as in the previous alignment. The trace however also contains activity `f` which is not included in the process model, causing a move on log only. Additionally, the trace contains activity `e` twice which also causes one move on log only. The alignment contains three move on

Table 5.2: Optimal alignments of the three traces of Figure 5.3b on the process model of Figure 5.5.

Trace	a	b	c	d	e	g
Model	a	b	c	d	e	g

(a) Alignment between the trace $\langle a, b, c, d, e, g \rangle$ (which occurs 80 times) and the process model, where the alignment cost is 0.

Trace	a	d	b	c	≫	e	g
Model	a	≫	b	c	d	e	g

(b) Alignment between the trace $\langle a, d, b, c, e, g \rangle$ (which occurs 5 times) and the process model, where the alignment cost is $5 + 2 = 7$.

Trace	a	b	d	c	≫	e	f	e	g
Model	a	b	≫	c	d	e	≫	≫	g

(c) Alignment between the trace $\langle a, b, d, c, e, f, e, g \rangle$ (which occurs 15 times) and the process model, where the alignment cost is $3 \times 5 + 2 = 17$.

log only and one move on model only, resulting in an overall cost of $3 \times 5 + 2 = 17$.

Given optimal alignments between the traces in an event log and the process model, the final replay fitness score is calculated as follows:

$$Q_{rf} = 1 - \frac{\sum_{\text{traces}} \text{cost for aligning model and trace} \times \text{trace frequency}}{\text{Cost to align log on model with no synchronous moves}} \tag{5.10}$$

where the denominator is the upper bound of the alignment cost for the optimal alignment. This is used to normalize the replay fitness to a value between 0 and 1.

It is important to note that the frequency of identical traces matter, hence frequently occurring traces have a bigger impact on the replay fitness than infrequent traces. Furthermore, calculating the cost to align a trace on a model without synchronous moves is trivial. These costs can be obtained by taking the length of the trace and the shortest run through the model. Obtaining the optimal alignment between a trace and the model however is the computationally expensive part.

Using the alignments shown in Table 5.2 we can calculate the replay fitness of the event log of Table 5.2 on the process model shown in Figure 5.5. The costs for each alignment are known and shown in Table 5.2. The 'cost to align log on model with no synchronous moves' consists of two parts. First the move on log only cost for all traces in the event log is calculated. The first trace occurs 80 times and contains 6 events, hence the move on log only costs are $80 \times 6 \times 5 =$

2,400. In a similar way the cost for the other two traces can be calculated. The cost for an execution of the process model without any synchronous moves can be calculated by finding the cost for the shortest path through the process model, and multiplying that with the number of traces in the event log. Since the event log contains 100 traces and the process model only allows for one execution sequence, which consists of executing six activities, these costs are $100 \times 6 \times 2 = 1{,}200$. Filling in Equation 5.10 with these values gives:

$$\begin{aligned} Q_{rf} &= 1 - \frac{80 \times 0 + 5 \times (5+2) + 15 \times (3 * 5 + 2)}{(80 \times 6 \times 5 + 5 \times 8 \times 5 + 15 \times 6 \times 5) + 100 \times 6 \times 2} \\ &= 1 - \frac{0 + 35 + 255}{(2{,}400 + 150 + 600) + 1{,}200} = 1 - \frac{290}{4{,}350} = 1 - 0.0667 = 0.9333 \end{aligned} \tag{5.11}$$

For the alignments as shown in Table 5.2, the resulting replay fitness therefore is 0.93.

Unfortunately, computing alignments is a complex task that violates the first requirement of efficient implementation. However, currently it is the most robust way of relating a process model with an event log. Moreover, the next two quality metrics use the information provided by the alignments without requiring additional complex computations. Using alignments, process models can also be more effectively repaired since an alignment indicates where a process model needs to be fixed (see Section 6.3).

Alignment based replay fitness provides intuitive results, since it directly uses the mismatches between event and activity execution, the notion of an alignment is easy to understand, and the way the final score is calculated can be clearly specified.

Selecting the best alignment between the traces and the process model is done by only considering the quality dimension of replay fitness. However, the choice of alignment directly influences the quality dimensions of precision and generalization, since they base their calculations on the alignments.

5.4.2 Other Replay Fitness Metrics

Replay fitness is the quality dimension that has attracted the most attention from researchers in the process mining field. In this section we discuss other replay fitness metrics that could be used. For a more complete overview of replay fitness metrics we refer to [21, 155].

Token-Based Replay

Token-based replay [155,157] takes the traces of an event log and replays them on a process model. The result of this approach applied on a Petri net version of the process model of Figure 5.5 results in the process model as shown in Figure 5.6. Since activity d was executed twenty times in the event log, before it actually could, twenty tokens are missing in the place before activity d. However, twenty tokens are also remaining since activity d is not executed after c occurs.

Although this approach indicates where problems exist in the process model, there are some disadvantages. The addition of missing tokens may allow for extra behavior that could not have been performed in the original Petri net. Especially in cases where deviations occur frequently, some places in the Petri net may be flooded, making the results less reliable. There are also issues with silent transitions and duplication of tasks. More important however is the assumption that the event log is always right. Unlike the alignment approach, events cannot be skipped in the token-based replay and only errors in the process model are considered. This results in the assumption that $L \subseteq S$, which is often not correct.

Replay with Artificial Negative Events

This approach applies the same idea as token-based replay, but now includes negative events [89,177]. In this approach negative events represent activities that could not have happend in the event log at specific locations in the trace. The approach also replays traces individually, and uses local heuristics to decide whether activities are enabled. This results in a computational complexity that is linear in the length of the traces. However, negative events are not provided in the event log and thus need to be derived. Since the real system is not known, only the event log can be used for this. The current approach introduces negative events to traces based on the other traces in the log. It is not known however whether the introduced negative events are really negative events, i.e., really capture behavior that is indeed not possible according to the system. This results in the assumption that $L = S$, i.e., that the event log demonstrates all

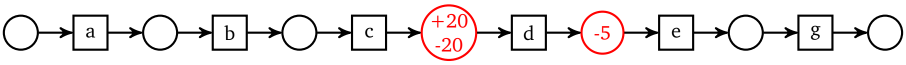


Figure 5.6: Example of a token-based replay result using the event log of Figure 5.3b and a Petri net version of the process model of Figure 5.5.

possible behavior and does not contain outliers.

Comparing Event Streams with Model Streams

An approach that compares event streams, i.e., traces, with process models to measure their similarity is proposed in [58] . Much like alignments, the similarity between the streams is measured by the fraction of insertions and deletions required to match the trace to a model stream. Several techniques are applied to avoid the well-known state-space explosion problem and reduce the overall complexity of the approach. However, none of these techniques guarantee that the result is the same as the one obtained without these techniques enabled. Moreover, the computational complexity is worse than that of alignments, while the results are the same at best. For a more detailed discussion of the differences between this approach and the alignment based replay fitness we refer to [21].

5.5 Precision

Precision indicates how much additional behavior the process model allows that is not seen in the event log. Precision therefore evaluates the following fraction:

$$\text{Model-log precision} = \frac{|L \cap M|}{|M|} \tag{5.1 repeated}$$

The problem with precision is that the behavior of the process model is potentially infinite in case of loops. Therefore estimates of the size of the allowed behavior of the process model need to be made.

The process model shown in Figure 5.7 is not very precise given the three observed traces. The process model allows for the execution of activities b, c and d in parallel. This allows for for six different traces to be produced ($\langle b,c,d\rangle$, $\langle b,d,c\rangle$, $\langle c,b,d\rangle$, $\langle c,d,b\rangle$, $\langle d,b,c\rangle$ and $\langle d,c,b\rangle$). Furthermore, e, f and g are children of a $\circlearrowleft$-construct, resulting in an infinite number of possible traces to be produced. The event log however only contains three different traces. Hence, the process model is not very precise.

However, enumerating all possible traces of the process model is not feasible for larger process models, especially when they have many parallel activities. Moreover, in case of loops, the allowed behavior is infinite. Therefore estimations of the allowed behavior of a process model need to be made.

5.5.1 Escaping Edges

By using the alignments calculated for the replay fitness dimension, and the state space constructed during these calculations, precision can be measured. An example is shown in Figure 5.8.

The state space shown in Figure 5.8 is constructed during the calculation of the alignments, where we only consider the process model behavior and thus ignore events skipped in the event log. The state space is not complete with respect to all allowed behavior of the process model. However, it does give insights into which parts of the state space have been used. [138] discusses how precision can be estimated by considering the escaping edges in a state space constructed during the calculations of alignments. Each escaping edge represents a decision in the process model that was possible but never made in the event log. If there are no escaping edges, precision is considered to be perfect.

Using the state space constructed by the alignment calculation, we calculate the precision as follows:

$$Q_p = 1 - \frac{\sum_{\text{visited markings}} \#\text{visits} \times (\#\text{outgoing edges} - \#\text{used edges by replay})}{\sum_{\text{visited markings}} \#\text{visits} \times \#\text{outgoing edges}} \qquad (5.12)$$

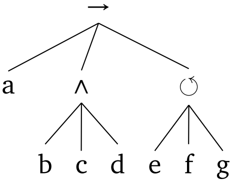


Figure 5.7: Example process model to explain precision

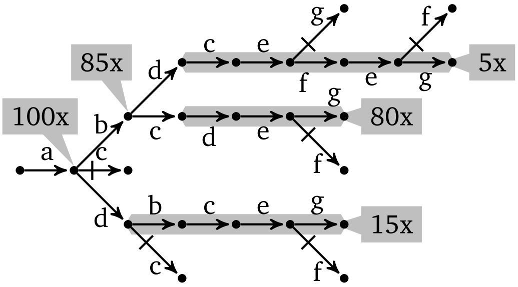


Figure 5.8: Partial state space as constructed during alignment calculation. Arrows with a dash indicate escaping edges.

For the example of Figure 5.8 this results in the following calculation:

$$
\begin{aligned}
Q_p =& 1 - \frac{\begin{array}{r} 100 \times (1-1) + 100 \times (3-2) + 85 \times (2-2) + 5 \times (1-1) + 5 \times (1-1) + 5 \times (2-1) \\ + 5 \times (1-1) + 5 \times (2-1) + 80 \times (1-1) + 80 \times (1-1) + 80 \times (2-1) \\ + 15 \times (2-1) + 15 \times (1-1) + 15 \times (1-1) + 15 \times (1-1) + 15 \times (2-1) \end{array}}{\begin{array}{r} 100 \times 1 + 100 \times 3 + 85 \times 2 + 5 \times 1 + 5 \times 1 + 5 \times 2 + 5 \times 1 + 5 \times 2 \\ + 80 \times 1 + 80 \times 1 + 80 \times 2 + 15 \times 2 + 15 \times 1 + 15 \times 1 + 15 \times 2 \end{array}} \\
=& 1 - \frac{0 + 100 + 0 + 0 + 0 + 5 + 0 + 5 + 0 + 0 + 80 + 15 + 0 + 0 + 0 + 15}{100 + 300 + 170 + 5 + 5 + 10 + 5 + 10 + 80 + 80 + 160 + 30 + 15 + 15 + 30} \\
=& 1 - \frac{220}{1{,}015} = 1 - 0.2167 = 0.7832
\end{aligned}
\tag{5.13}
$$

For the example of Figure 5.8 the precision is 0.78.

The requirements for evaluation metrics are all satisfied. This metric is efficient, since it reuses information from the alignment metric without significant overhead. Results of this metric are also intuitive and the specification is clear since the metric uses the notion of escaping edges, which relates to unused behavior. Finally, this metric is also orthogonal to the other quality dimensions. Although we reuse information obtained by the alignment metric, we measure only the precision dimension.

5.5.2 Other Precision Metrics

Other precision metrics exist, although not as many as for the replay fitness quality dimension. In this section we discuss two other metrics: advanced behavioral appropriateness and the behavioral specificity metric.

Advanced Behavioral Appropriateness

The advanced behavioral appropriateness metric is an improvement of the 'basic' behavioral appropriateness metric, both of which are presented in [157]. The advanced behavioral appropriateness metric compares pairwise relations between activities in the event log to the pairwise relations of activities in the process model. However, the computation of all these pairwise relations is expensive and therefore not feasible in practice. In [27] a different precision metric is proposed that utilizes the total number of enabled activities in all visited states of the process model during replay, using token-based replay. Since missing tokens might be added when necessary for replay, this metric shows misleading results for non-perfectly fitting traces: the Petri net is flooded, so tokens enable an unreasonable number of transitions.

Behavioral Specificity

The behavioral specificity metric uses artificial negative events to measure precision. It therefore uses the approach as discussed in Section 5.4.2 and [89]. This metric reduces the precision score of a process model whenever negative events were enabled while replaying the positive events in the event log on the process model. The main limitation of this approach is the assumption that the event log is a complete observation of all allowed behavior of the system, i.e., $L = S$.

5.6 Generalization

Generalization estimates how well the process model describes the behavior of the (unknown) system, and not only the event log with the observed system behavior. Recall that generalization corresponds to the following fraction:

$$\text{Model-system recall} = \frac{|S \cap M|}{|S|} \qquad \text{(5.6 revisited)}$$

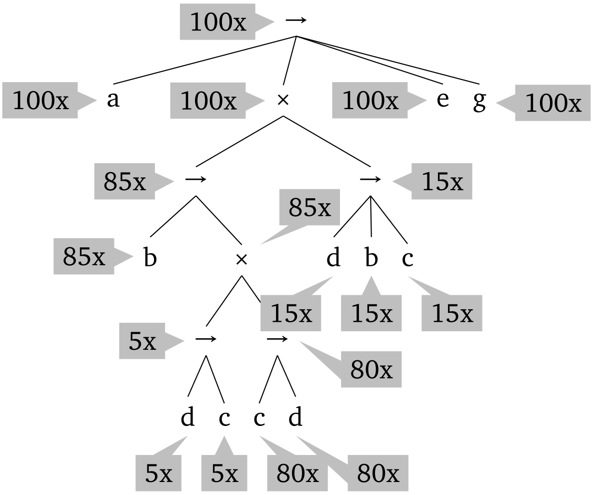


Figure 5.9: Example process model which is not general.

Figure 5.9 shows another process model that can be used to explain the observed behavior of the event log in Figure 5.3b. Although this process model describes the observed behavior correctly and precisely, it is not as generic as it should be. It is therefore likely that the process model does not give insights into the behavior of the system, but instead only describes the observed example behavior of the event log.

However, since the behavior of the system is not known, the generalization of a process model needs to be evaluated using derived metrics.

5.6.1 Frequency of Use

If all parts of the process model are frequently used, the process model is likely to be generic. However, if some parts of the process model are rarely used, chances are high that the system actually allows for more behavior, and that the process model is describing certain behavior in an overly precise and specific way. Therefore we base the generalization metric on how often nodes of the process tree have been visited while replaying the event log. For this we use the alignment provided by the replay fitness algorithm. The more a node is visited, the more certain we are about the statistics obtained via replay, i.e., if the node explains the observed behavior well of not. If some parts of the tree are visited very infrequently, generalization is poor. Therefore, generalization is calculated as follows:

$$Q_g = 1 - \frac{\sum_{\text{nodes}}(\sqrt{\#\text{executions}})^{-1}}{\#\text{nodes in model}} \tag{5.14}$$

The square root of the number of executions is taken because the effect of having 10 executions instead of 1 is considered a more significant improvement than going from 10 to 100 executions. From each of these values the power of -1 is taken to normalize it to a value between 0 and 1. Then these values are summed and divided by the total number of nodes in the tree to get the average for the whole tree. Please note that, unlike the proposed metrics for simplicity, replay fitness and precision, this metric can only reach the value of 1 in the limit (because the fraction can never be 0).

However, this equation has one major disadvantage: it contains a loophole that is used by the ETM framework to (artificially) increase this metric. The ETM framework introduces a lot of silent steps and useless loops to increase the number of nodes that are executed. Therefore, we incorporate the notion of useless nodes from Section 5.3.1 into this metric as follows:

$$Q_g = 1 - \frac{\sum_{\text{non-useless nodes}} (\sqrt{\#\text{executions}})^{-1}}{\#\text{nodes in model}} \tag{5.15}$$

That way, even if simplicity is evaluated using a different metric, only nodes that actually contribute to the process model are evaluated by the ETM framework to determine generalization.

In the example process model of Figure 5.9 the number of times each transition has been executed in the process model is indicated. When using these frequencies to fill in Equation 5.15 this results in a generalization of:

$$\begin{aligned} Q_g &= 1 - \frac{\begin{array}{r} \sqrt{100}^{-1} + \sqrt{100}^{-1} + \sqrt{100}^{-1} + \sqrt{85}^{-1} + \sqrt{85}^{-1} + \sqrt{85}^{-1} \\ + \sqrt{5}^{-1} + \sqrt{5}^{-1} + \sqrt{5}^{-1} + \sqrt{80}^{-1} + \sqrt{80}^{-1} + \sqrt{80}^{-1} \\ + \sqrt{15}^{-1} + \sqrt{15}^{-1} + \sqrt{15}^{-1} + \sqrt{15}^{-1} + \sqrt{100}^{-1} + \sqrt{100}^{-1} \end{array}}{18} \\ &= 1 - \frac{3 \times 0.1 + 3 \times 0.1085 + 3 \times 0.4472 + 3 \times 0.1118 + 4 \times 0.2582 + 2 \times 0.1}{18} \\ &= 1 - \frac{3.5352}{18} = 1 - 0.1964 = 0.80 \end{aligned} \tag{5.16}$$

Most of the requirements for evaluation metrics are satisfied for the generalization metric of frequency of use. This metric is very efficient, since it reuses information from the alignment metric without significant overhead. Results of this metric are also intuitive and the specification is clear. Finally, this metric is not fully orthogonal to other quality dimensions, since it incorporates the notion of useless nodes from the corresponding simplicity metric. This is however necessary to prevent this metric from only functioning in combination with this particular simplicity metric.

5.6.2 Other Generalization Metrics

Not many metrics for generalization currently exist, although it can be easily shown that this dimension is crucial for evaluating the quality of a process model, since replay fitness and precision alone are not enough.

Change of a New Observation

One of a few generalization metrics is presented in [10] which uses an estimator from statistics [39] that estimates the chance of a new observation, using the total number of observations and the number of unique observations. The reasoning is that if each observation was a unique one, the chance that the next observation will be unique is high. However, if there were many observations of only a few unique ones, chances are small that a next observation will be unique. By applying this to state visits, so by calculating the probability that a next visit to a state will reveal a path not seen before, generalization can be estimated. However, this approach does not work for parallelism, since each trace might be a unique sequence of activities, while the process model is actually general. The same idea could be applied to the process tree itself, by evaluating in how many different ways an operator behaved. In this case the problem lies in the fact that parallel behavior should be abstracted from, to prevent the same problem as for the state space. At the same time, if only few actual sequences are observed, an $\wedge$ or $\vee$-operator is generalizing too much.

Advanced Behavioral Appropriateness

The advanced behavioral appropriateness metric of [157] also includes generalization, since it compares the behavior of the process model and the event log in both directions. However, here generalization is seen as the inverse of precision, i.e., if a model is less precise it automatically becomes more general. The concept of an unknown system is not incorporated in this metric.

Behavioral Recall and Causal Footprint

Other approaches such as behavioral recall [27] and the causal footprint metric [68, 69] are able to quantify how general a given model is by using a reference model that serves as the system. However, the original model is not known in the setting of process discovery, and these techniques are therefore not applicable to our setting.

K-fold Cross Validation

A common way in data mining to evaluate how good a classification algorithm is, is to use a so-called holdout method, e.g. k-fold cross validation [137]. The data set is split in *k* parts and *k-1* parts are used for training the algorithm, while the k^{th} part is used for evaluating the performance of the algorithm. Generalization however is not about evaluating the performance (or generalization) of the algorithm, but of the discovered process model with respect to the system.

5.7 The Importance of Considering all Four Quality Dimensions

Although most process discovery algorithms only consider one or two quality dimensions, all four should always be considered. The process models used to explain each of the quality dimensions are repeated in Figure 5.10.

The process model used to explain simplicity is shown in Figure 5.10a. This process tree can replay all traces, is very precise and general but is not as simple as it could be. The process tree of Figure 5.10b is very precise, general and simple, but scores bad on replay fitness. Since replay fitness provides the relation between the process model and the event log, a low score on replay fitness means that the process model is not really related to the observed behavior. Therefore, the quality dimensions of precision, generalization and simplicity are hard to interpret when replay fitness is low.

The other two process models also score well in three out of the four quality dimensions, but score badly in the remaining quality dimension. The overall best process model is the process model of Figure 5.3: it is the simplest process model that is able to replay all behavior, does not allow for additional behavior and describes the process in a general way.

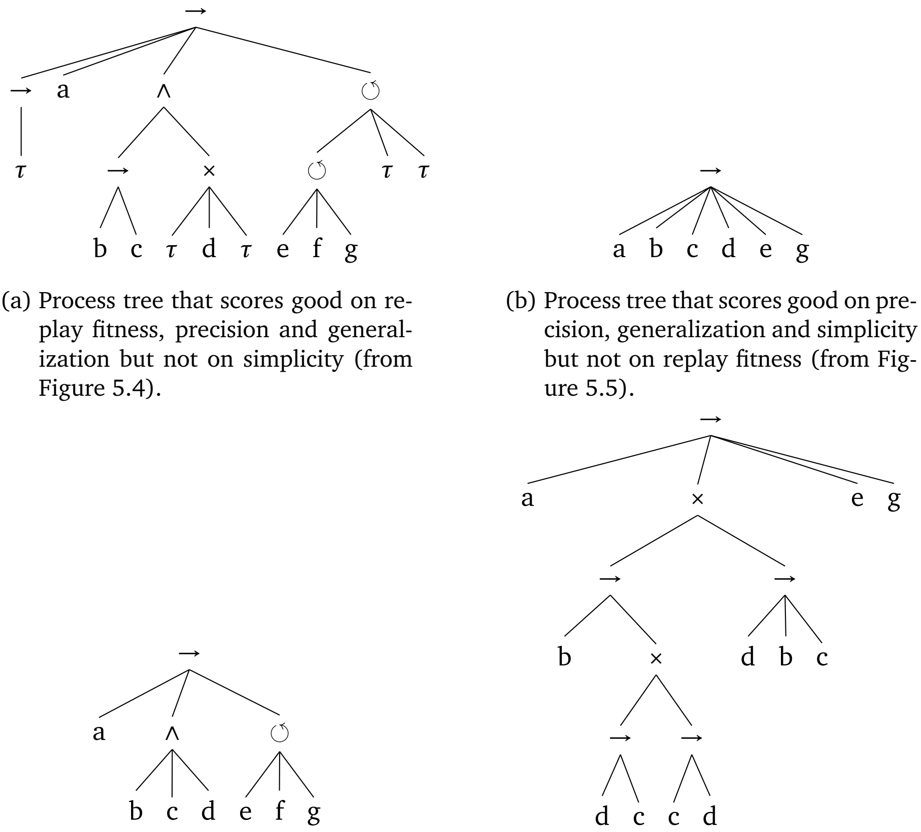
Figure 5.10 illustrates that all four quality dimensions need to be considered, and that they are orthogonal to each other. However, the quality dimension of replay fitness is the most important, since a low replay fitness means that the process model has little relation to the observed behavior. Unfortunately none of the current process discovery algorithms incorporates all four quality dimensions.

5.8 Quality Metric Considerations

In this chapter we proposed a metric for each of the four quality dimensions such that the ETM framework can incorporate each dimension in the evaluation of candidates. The four quality dimensions are proposed in previous literature, mainly [5], and their correctness and completeness have been shown by the discussion based on the Venn diagram of Figure 5.2.

Defining or choosing the metrics used to evaluate a quality dimension is not straightforward and multiple options exist for each quality dimension. In this chapter we have proposed metrics for the quality dimensions for use within the ETM framework. However, different metrics might be more suitable for different situations.

For evaluating replay fitness for instance one could also count the fraction of traces can be fully replayed in the process model. In certain scenarios this might be a better metric than the more detailed alignment calculations. This is



(a) Process tree that scores good on replay fitness, precision and generalization but not on simplicity (from Figure 5.4).

(b) Process tree that scores good on precision, generalization and simplicity but not on replay fitness (from Figure 5.5).

(c) Process tree that scores good on replay fitness, generalization and simplicity but not on precision (from Figure 5.7).

(d) Process tree that scores good on replay fitness, precision and simplicity but not on generalization (from Figure 5.9).

Figure 5.10: Four process models that each score well on three of the four quality dimensions, but none of them are sufficiently describing the system.

also true for metrics for the other quality dimensions. However, it is important to also consider the chosen metrics together, since they should work well together. Therefore, the choice of the four quality metrics used in the remainder of this thesis is a delicate one. And, as is shown in Section 5.7, together these four quality metrics cover the four quality dimensions and all four metrics are required.

We do not claim however that these four quality metrics are always the best or only choice. Therefore the ETM framework is flexible in which specific metrics are used during the evaluation. We show in the remainder of this thesis, especially in Section 6.8 and Chapter 7, that the chosen combination of the four quality metrics works well in practice.

5.9 Additional Quality Dimensions

Although we showed that the four quality dimensions presented here are the minimal quality dimensions to consider, additional quality dimensions can be considered. As discussed in the scenarios in Section 4.2, other aspects can be considered during process discovery. Consider for instance the similarity to a given process model, or the quality of the configurations present in the discovered configurable process model. Additional quality dimensions can be used to evaluate other aspects of the process model such as the data or resource perspectives. There is no upper limit to the number of quality dimensions that can be used, but the four quality dimensions discussed in this chapter form a minimal set of quality dimensions to consider. It is important however that the requirements as discussed in Section 4.3.4 are taken into consideration when implementing additional quality dimensions.

5.10 Related Work

Process model quality is expressed by using the four quality dimensions of *replay fitness*, *precision*, *generalization* and *simplicity* [5,8,10]. For each of these quality dimensions several implementations exist.

The most robust metric for replay fitness to date is presented in [10, 21, 22] where alignments are calculated between traces of the event log and executions of the process model This approach finds an alignment with the lowest cost, where deviations are assigned costs. Many other metrics for replay fitness exist such as token-based replay [155, 157], artificial negative events [89, 177] and comparing event streams with model streams [58]. For a more complete overview of different replay fitness metrics we refer to [21, 155].

Precision quantifies the fraction of unseen behavior modeled by the process model. However, loops in the process model can generate possibly infinite behavior, complicating the computation of this quality dimension. A solution is proposed in [138] where the state space as constructed during the calculation of the alignments for replay fitness is used. The notion of unused or escaping edges quantifies the number of unobserved branches in the state space. Other precision metrics based on token based replay [27, 157] and negative events [89] are suggested, which suffer from the same issues as the metrics they are based upon.

The quality dimension of generalization is the most difficult quality dimension to asses. A metric from statistics is applied in [10] to evaluate the diversity of a population. However, issues arise in the case of parallelism. In [157] a generalization metric is proposed but assumed to be the inverse of precision, which we argue it is not. Additional generalization metrics are proposed in [27,68,69] that require a given process model to represent the system. However, we argue that the system is usually unknown and generalization should be calculated without concrete knowledge of the system.

The fourth quality dimension of simplicity is a quality dimension that is not directly related to the observed behavior. In [133] different simplicity (or complexity) metrics, and their relation to errors in process models, are extensively discussed. The main observation is that size is the most important measure for simplicity.

5.11 Conclusion

In this chapter we positioned the four well-known quality dimensions of simplicity, replay fitness, precision and generalization. By explicitly considering the (unknown) system we provided insights into the interplay between the behavior of the system, the observed behavior in the event log and the possible behavior of the process model. Additionally we explained each quality dimension using a simple example. This helps in understanding the quality of a process model given an event log, as is described as Challenge 4 in Section 1.3. In this chapter we also discussed several possible ways to measure each of these four quality dimensions. Finally, we showed that all four quality dimensions are really necessary to evaluate the quality of a process model.

In the next chapter we implement a process discovery algorithm based on the ETM framework: the ETM*d* algorithm. The quality dimensions discussed in this chapter are used by the ETM*d* algorithm to discover process trees from several artificial and real life data sets in Chapter 7. In Chapter 8 and Chapter 9 we discuss the ETM*r* and ETM*c* algorithms respectively which use additional quality dimensions.

Chapter 6

Discovery of Process Trees

In this chapter we discuss the implementation of the ETM*d* algorithm based on the ETM framework as discussed in Section 4.1 and shown in Figure 6.1. We start with a discussion on the three different types of change operations defined for evolutionary algorithms: creation of the initial population, mutation and crossover. The creation of the initial population is discussed in Section 6.1, followed by a selection of mutation operations, which is divided into random mutations, discussed in Section 6.2, and guided mutations which is discussed in Section 6.3. The third and last change operation, which we discuss in Section 6.4, is crossover. The next element to be detailed is that of candidate selection, which is discussed in Section 6.5. The final element in the ETM*d* algorithm, termination condition(s), is discussed in Section 6.6.

The ETM*d* algorithm is applied on a running example in Section 6.8, where besides the best single process tree, a Pareto front is also discovered. Section 6.9 discusses the results of current state-of-the-art process discovery algorithms on the same running example. Section 6.10 concludes this chapter.

6.1 Initial Population Creation

As discussed in Chapter 4 the first step of the ETM*d* algorithm is the creation of new process trees to form the initial population. Additionally, in each generation, some process trees might be selected to be replaced by newly created trees.

As mentioned in Section 4.4.3, it is important to note that the creation of new trees does not imply that the initially created process trees should be of good quality. The evolutionary character of the ETM framework provides the freedom to construct less-than-optimal process models. The main purpose of initial population creation is initializing the ETM*d* algorithm with a diverse set of candidates. The change operations of mutation and crossover are used to (possibly) further improve the quality of the candidates according to the used quality dimensions. Therefore population creation should mainly focus on the exploration aspect of the evolutionary algorithm, and leave the exploitation to the other change operations.

In this section we discuss different approaches for the creation of process trees: (1) random process tree creation, (2) advanced trace-model creation and (3) creation using existing process discovery algorithms. The random approach is discussed in Section 6.1.1. This approach almost entirely ignores the information recorded in the event log. In Section 6.1.2 we discuss a smarter approach that creates process trees using traces from the event log. The resulting process trees are not perfect but are very suitable for the (guided) mutation operators we define later in this chapter. Finally, in Section 6.1.3 we discuss the possibility of using other process discovery techniques to build initial process trees.

It is important to note that all these different 'tactics' are required because beforehand it is not known which quality dimensions are included, and what their relative importance is. Therefore we cannot fully rely on constructive methods to create the initial population.

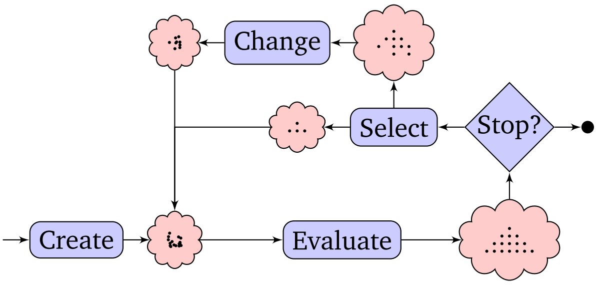


Figure 6.1: The basic framework for evolutionary process discovery.

6.1.1 Random Tree Creation

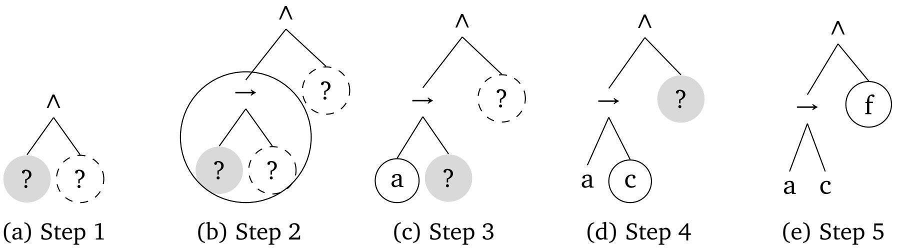
One of the simplest methods to create new process trees is by randomly constructing them. An example of the process of created a random process tree is shown in Figure 6.2. Using the activities of the event log, the process tree is randomly built from the root. Each node is assigned at random to represent either an activity or an operator node. The first step of this process is shown in Figure 6.2a. If the node is selected to be an operator, the type of operator is selected by chance, where some operators might have a higher probability of being selected. In this case the root is chosen to be an ∧-operator. The left-most child of this operator is again an operator, of type →, as is shown in Figure 6.2b. If a node represents an activity, this activity is randomly selected from the list of activities present in the event log. This is shown in Figure 6.2c, Figure 6.2d and Figure 6.2e.

6.1.2 Using Advanced Trace-Model Creation

Another way of creating new process trees is by first creating process trees that describe individual traces, and subsequently merging these process models.

Creating trace models

In this step a process tree is created that describes a single trace. If a trace contains each activity exactly once, building a process tree can simply be done by placing the activities underneath a →-operator, in the same order as the



(a) Step 1 (b) Step 2 (c) Step 3 (d) Step 4 (e) Step 5

Figure 6.2: Example of random construction of a process tree. A newly added node is encircled, the currently selected node to be set has a gray background and remaining empty places in the process tree have a dashed circle.

activities are encountered in the trace. This type of tree is called a *trace-model* [74,75].

In case the trace contains activities multiple times, a ↺-operator can be inserted in the process tree. An example of a trace-model with duplicate activities, for the trace $\langle a,b,c,d,e,b,d,e,f,g\rangle$, is shown in Figure 6.3. Determining which activities need to be assigned to the ↺-operator is done in two phases. In the first phase the activities that are duplicated are detected by counting how often each activity occurs. In the example of Figure 6.3 activities b, d and e are duplicated, as is indicated by the gray rectangles in the process tree. In the next phase the trace-model is again parsed, and at the point where the first activity that occurs multiple times is encountered, an ↺-node is inserted. For the example of Figure 6.3 the ↺-node is inserted after activity a, as is shown by the cloud symbol in the top right tree. Each activity that occurs multiple times is assigned randomly to either the 'do' or 'redo' part of the ↺-operator. The order of the activities within each of these parts is maintained. Hence, the ↺ of Figure 6.3 contains activities b, d and e. As is shown at the bottom of the figure, in total eight distributions of these three activities are possible over the different 'do' and 'redo' parts of the ↺-operator. All other occurrences of the activities placed under the ↺-operator are removed from the trace-model. This ensures that the

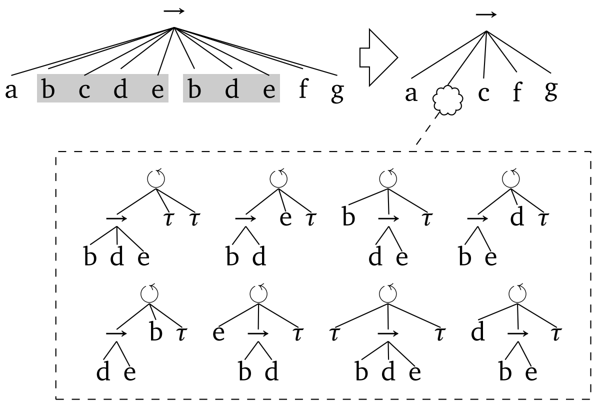


Figure 6.3: Example of advanced trace-model creation for the trace $\langle a,b,c,d,e,b,d,e,f,g\rangle$. The top left model shows the (basic) trace model with the regions of duplicate activities marked. The advanced trace model (shown at the top right), contains each activity once. The cloud in the top right model is replaced with one of the possible ↺-sub trees.

duplicate activities appear only once in the resulting process tree.

Merging of the initial models

The next step of this approach is merging the created process trees into process trees that describe multiple traces. Traces are selected randomly from the event log, and their corresponding trace-models are merged. This process stops when all selected traces have been processed. Moreover, since duplication of activities in a process tree allows for multiple mappings between two process trees, the process also stops when activity duplications are introduced in a process tree.

The first step when merging two process trees is to create a mapping between the nodes of the process trees [74, 75]. This can be done in a relatively straight-forward way since the process trees contain no duplicate activities. The order of activities is ignored, and unmapped leaves of either one of the input trees are copied into the merged process tree. An example is shown in Figure 6.4, where three trace models are merged in two steps. For the first two models on the left all leaves can be mapped, except `e` (marked gray) in the upper model of Figure 6.4. The location in the merged tree is determined by finding the location of the predecessor and successor of this node. In this case the location is determined to be between `d` and `f` in both trees. At this location an ×-node with `e` and τ as children is inserted, since in one model `e` is present while in the other model no activity is present at that location.

The resulting process model is then merged with a third process model for the trace $\langle a, c, b, h, i, g \rangle$. In the middle model the part between activities `c` and `g` is different (marked with dashed boxes), while in the right model activities `h` and `i` are different (marked with dashed circles). These two parts are placed underneath an ×-operator in the merged process tree.

The approaches for trace-model creation and merging these models follow a simple and straight-forward approach. Since the goal is to quickly create reasonable process trees, no advanced and time consuming calculations are made. Different process trees are created that describe the observed behavior. In a later phase these models are improved by the ETM*d* algorithm. The models created by using these methods are tailored for easy improvement by the guided mutation operators as discussed in Section 6.3.

6.1.3 Using Other Process Discovery Algorithms

Another way to create process trees is by using the result of other process discovery algorithms. However, most other algorithms produce process models in process modeling formalisms that cannot always be translated to process trees (for a more detailed discussion we refer to Section 3.5.2). One notable exception is the Inductive Miner (IM) [120, 121] that also produces process trees. By changing some of the parameters of the IM algorithm, different process trees can be obtained using the same event log.

However, the IM algorithm has some disadvantages the ETM*d* algorithm does not have. First of all the IM is relatively sensitive to both exceptional behavior and incompleteness of the event log, although extensions to combat this are proposed [121]. Furthermore, in case the IM cannot find a strong relation in part of the event log, it falls back to the flower model for that part of the subtree. Within a flower model all activities are able to be executed zero or more times. This therefore results in one or more ↻-operators with the activities of that part of the event log in a ×-construct in the 'do' part.

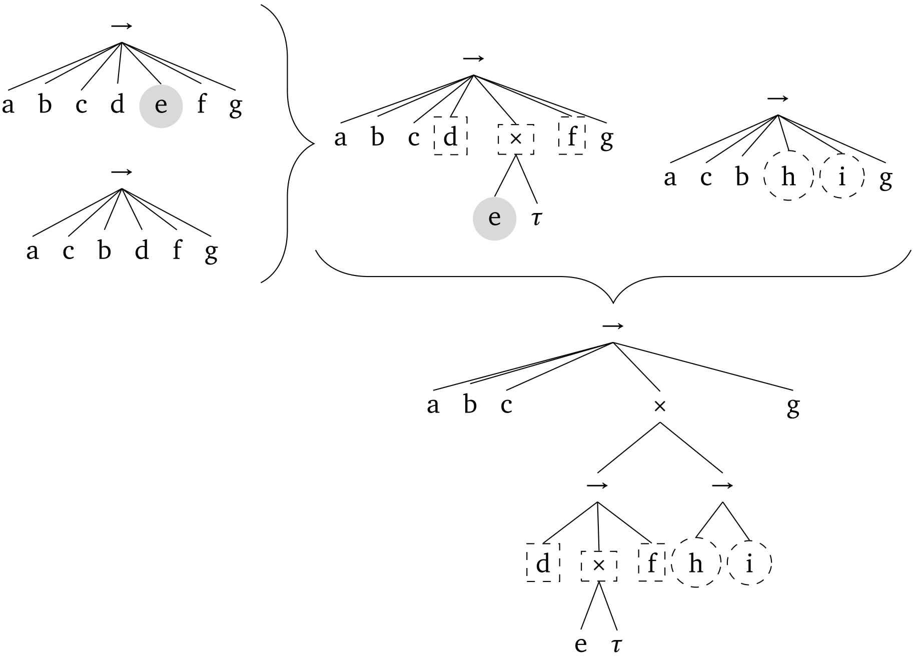


Figure 6.4: Merging of trace models with nodes in the original and merged model marked for clarity.

Although the results of the IM are of relatively good quality if the event log has no exceptional behavior and is complete, the ETM*d* algorithm is still needed to make further improvements. For instance, the IM algorithm cannot handle duplicate activities, i.e., activities with the same label that appear in more than one location in the process model. The ETM*d* algorithm does not have this issue in general, since most of the change operations can introduce activities into the process tree, regardless of whether they are already included.

Moreover, the IM algorithm uses a fixed approach to discover a process tree. This causes the resulting process trees to have a specific balance of the different quality dimensions, where mainly the dimensions of replay fitness and precision are considered. Additional quality dimensions, such as similarity (which we discuss in more detail in Chapter 8), cannot be considered by the IM algorithm. Therefore, the ETM*d* algorithm is still required to further improve the process tree according to the quality dimensions currently set. Nonetheless, the process trees produced by the IM algorithm might have a better quality than process trees created by the other initial population creation approaches.

In the remainder of this thesis *the IM algorithm is not enabled* in the ETM*d* (and derived) algorithms since this would complicate the discussion regarding the capabilities of the ETM framework and implemented algorithms. In practice however, enabling the IM algorithm will result in quicker, but not necessarily better, results.

6.2 Random Mutation

As discussed in Section 4.4.3 one of the change operations in an evolutionary algorithm is mutation. Mutations take a single process tree and modify (parts of) it. In general, three different mutation types can be identified for nodes in trees:

1. Removal of nodes;
2. Addition of nodes;
3. Updating of nodes.

In theory only the first two mutation types are necessary, since by removing and adding in sequence, all possible changes can be applied. However, since the overall quality of the process tree can decrease during this sequence of operations, we also introduce mutation operations that update the nodes in the process tree, for instance by changing the operator type of a particular node.

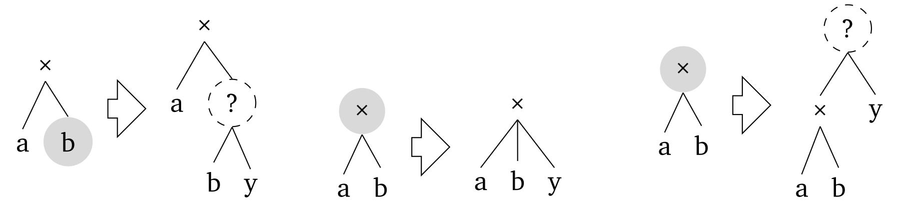
Some of these mutations do not change the behavior described by the process tree but only change the structure of the process tree. These operations are required to restructure the process tree so that future mutation operations can be performed with a better result. Consider for instance reordering the children of an ∧-operator such that this operator can be changed to an →-operator in a next generation.

Several random mutation operators can be defined that add, remove or update nodes:

Random node removal randomly selects a node from the process tree and removes it from the process tree, except when its parent is an ↺-node. In general applying this mutation results in removing behavior.

Random node addition randomly selects a node in the process tree and randomly selects an activity from the event log. Three different situations can be encountered, as is shown in Figure 6.5:

1. If the selected node in the tree is a leaf, as is shown in Figure 6.5a, an operator node is randomly chosen. The selected node and the randomly chosen activity are then placed in the tree at the location of the selected node under the randomly chosen operator.
2. If the selected node is an operator node the activity can be added as an additional child, in an arbitrary location, to this operator node. This situation is shown in Figure 6.5b.
3. It is also possible to add the activity next to the selected operator node, under a randomly chosen operator, as is shown in Figure 6.5c.



(a) New operator child added.

(b) New child added.

(c) New parent operator added.

Figure 6.5: Three types of node addition mutation where leaf y is added.

In case the selected operator is of the type $\circlearrowleft$, another node in the process tree is selected instead.

Random node mutation randomly selects a node from the process tree and changes its type. If the node is a leaf it is assigned a new activity. In case the node is an operator, it is assigned another operator type, except the $\circlearrowleft$-operator (because of the strict 3 children requirement of the $\circlearrowleft$-operator).

Normalization mutation applies a normalization operation on the whole process tree. Normalization consists of two phases: flattening and sorting. Flattening of a process tree means that operators that have children that are of the same operator type are ‘flattened’ by absorbing the children of the child-operator, except when the node is an $\circlearrowleft$-operator, since flattening can change the behavior. The sorting phase sorts the children of a node alphabetically (in case of leaves), by operator type and then by size of the subtree. Sorting is not applied to $\rightarrow$ and $\circlearrowleft$ operators since this changes the behavior.

Remove useless node mutation randomly selects a useless node from the tree (if there are any) and removes it without changing the behavior of the process tree. Useless nodes, as defined in Section 5.3, are nodes that do not add to or restrict the behavior in a process tree, and therefore can be removed without changing the behavior of the process tree. For example a τ in a $\rightarrow$-construct or $\wedge$-construct can simply be removed, without changing the behavior of the process tree.

Replace tree is a rather aggressive mutation operator which replaces the whole tree by a randomly created process tree. Here the approaches discussed in Section 6.1 for the creation of the initial population are used. The main purpose of this mutation operation is to increase the variation of the population.

Shuffling nodes rearranges the order of the children in a $\times$-operator, $\wedge$-operator and $\vee$-operator, selected at random from the process tree. The main purpose of this mutation is to change the structure of the process tree to possibly make it better suited for one of the other mutations to improve the quality of the process tree.

Although the random mutation operations discussed in this section together in theory allow for all possible process trees to be created, more directed change operations help in finding good quality process trees quicker.

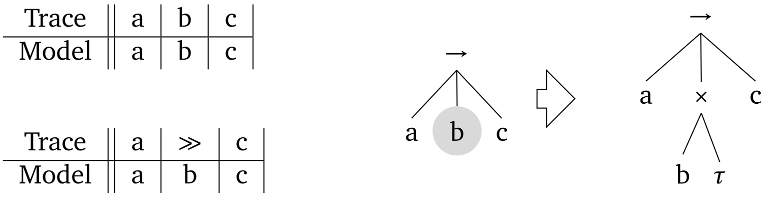
6.3 Guided Mutation

In the previous section examples of random mutations were discussed. Although these mutations together allow for any process tree to be discovered eventually, more targeted, or *guided mutation operators* can speed up the search process. However, most guided mutations require additional information to be able to apply directed changes to the process tree. Information that relates the process tree to the behavior of the event log is stored in the alignments, used to calculate the replay fitness quality, as is explained in Section 5.4.1. Similar to random mutation, three different mutation types can be distinguished: removing behavior, adding behavior and changing behavior.

The type of mutation to be applied is determined by first randomly selecting a node from the process tree. Using the information from the alignments the observed behavior is aligned with the behavior as described in the process tree [74, 75]. If the selected node is a leaf, and this leaf is mainly skipped during replay (i.e., mainly move on model only), then behavior regarding that leaf is removed, which is discussed in Section 6.3.1. If no move on model only is observed, then the observed behavior in the event log (i.e., move on log only) is used to add behavior to the process tree, which is discussed in Section 6.3.2. In the other case, when the selected node is an operator node, the behavior of that subtree is changed, which is discussed in Section 6.3.3.

6.3.1 Removing Behavior

Removing behavior from process trees can be done by removing leaves or by making parts of the process tree skippable [74, 75]. Which of these two is best depends on the behavior shown in the entire event log. Consider for instance the process tree of Figure 6.6 and an event log consisting of the two traces



Trace	a	b	c
Model	a	b	c

Trace	a	≫	c
Model	a	b	c

Figure 6.6: Removing behavior: leaf b is selected, and based on the traces $\langle a, b, c\rangle$ and $\langle a, c\rangle$, allowed to be skipped.

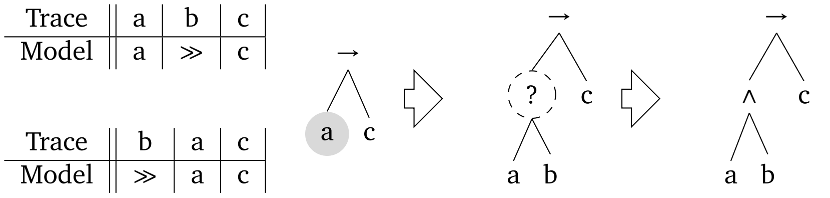
$\langle a, b, c\rangle$ and $\langle a, c\rangle$. The trace $\langle a, b, c\rangle$ can be aligned to the process tree perfectly. For the trace $\langle a, c\rangle$ the model has to make a move on model only on leaf b. Using these two alignments the process tree can be repaired by allowing leaf b to be skipped. This is achieved by adding an ×-operator as a new parent of b, and adding a τ-node as the other child of the ×-operator.

Another example would be when activity b was never observed in this location of the process tree, i.e., leaf b would always be a move on model only, and never be moved synchronously with an event from a trace. In this case leaf b can be removed to improve the process tree. Since ↺-operators require exactly three children, the leaf is replaced with a τ leaf if the parent is an ↺-operator.

6.3.2 Adding Behavior

Instead of observing mainly moves on model only, moves on log only can also be frequently observed. In this case behavior, i.e., a leaf with the corresponding activity, has to be added to better explain the observed behavior [74, 75]. Consider for example the process tree of Figure 6.7, with the observed traces $\langle a, b, c\rangle$ and $\langle b, a, c\rangle$. Since activity b is not present in the process tree, the move on log only observations are related to nodes in the process tree just before and after the observed move on log only in the alignment. Currently node a is selected for mutation, and to this node the log moves for activity b are also related. This indicates that at this location in the process tree activity b needs to be added. Therefore, an operator is added as a new parent of a, and b is also added to this operator. Next, the type of operator needs to be determined. This is done by investigating the behavior of activities a and b in the alignments.

Determining the operator that best describes a pair of activities is rather straight-forward by inspecting how they occur together in the traces [74, 75]. In case the activities are always observed in a certain order, an →-operator



Trace	a	b	c
Model	a	≫	c

Trace	b	a	c
Model	≫	a	c

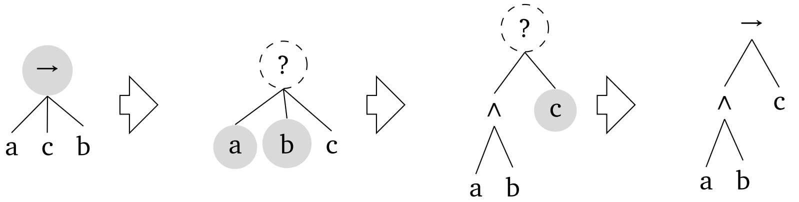
Figure 6.7: Adding behavior: leaf a is selected, and based on the traces $\langle a, b, c\rangle$ and $\langle b, a, c\rangle$, activity b is added under an ∧-operator.

is added, with the two activities in the correct order. If no particular order is observed an ∧-operator is used. If only one of each activities is observed in a trace at the time, an ×-operator is selected. In case sometimes one and sometimes both activities are observed, an ∨-operator is used. Finally, in case activities are observed multiple times the ↺-operator is added.

6.3.3 Changing Behavior

In case an operator type has been randomly selected in the beginning (see the introduction of this section), the whole subtree under that operator node is rebuilt [74, 75]. This is done by randomly selecting two activities from the subtree, which are then joined using the operator type as detected using the patterns described in Section 6.3.2. As long as there are unprocessed activities from the original subtree, the next activity is selected. The operator type that best explains the relation between the selected activity and the activities in the current tree is again determined and added to the process tree. This is repeated until all activities of the selected operator are processed.

This processes is demonstrated in Figure 6.8 where in the leftmost tree the root is selected for guided mutation. The tree is rebuilt based on the traces $\langle a, b, c\rangle$ and $\langle b, a, c\rangle$. In the first step, activities a and b are selected, for which the ∧-operator is detected as is shown by the third process tree. Finally, activity c is added to this subtree using an →-operator.



Trace	a	b	c	≫
Model	a	≫	c	b

Trace	b	a	c	≫
Model	≫	a	c	b

Figure 6.8: Changing behavior: the root operator → selected, and based on the traces $\langle a, b, c\rangle$ and $\langle b, a, c\rangle$, the whole sub-tree is changed.

6.4 Crossover

Crossover is a common operator in evolutionary algorithms and is inspired by the mating process of species [77]. However, crossover is not beneficial in all algorithm scenarios [163]. We have found through experimentation [74] that crossover does not contribute to the efficiency of the ETM*d* algorithm. The main cause is that it is very unlikely that parts of two process trees together explain the observed behavior better than either one of the original process trees. Therefore, although a basic crossover operation is available in the ETM*d* algorithm, we apply it with low frequency and rely more on mutation.

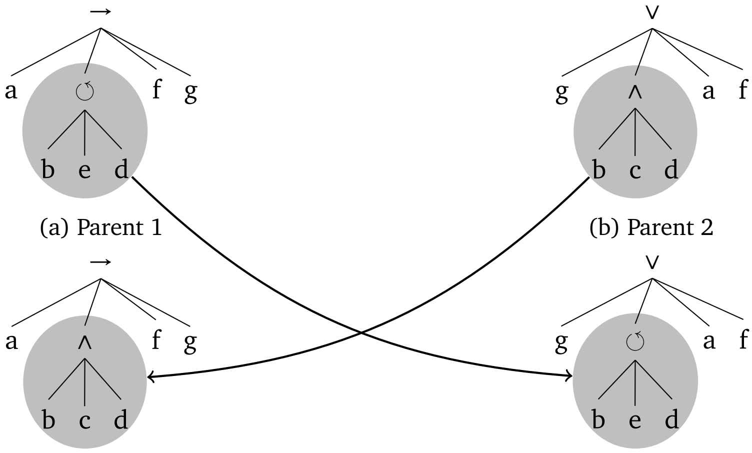
In essence, a crossover operator takes two candidate solutions and creates offspring by swapping parts between the selected candidates. An example is shown in Figure 6.9, where in each parent one node is selected and swapped to create the offspring. The created offspring has the same overall structure as one of the parents, but the selected node is replaced by that of the other parent. The main challenge for crossover is to select the correct locations in both trees such that applying crossover improves the overall quality of at least one of the offspring with respect to the parents. A basic crossover implementation randomly selects parts in each parent that are to be swapped. A more advanced approach to select parts has been discussed and evaluated in [74] but did not result in a more effective search by the ETM*d* algorithm.

The general idea of crossover is that both parents have good sections describing different parts of the process [77], i.e., one parent may have captured the first half of the process well, whereas the other parent is better at capturing the second half. The offspring created by crossover is hoped to combine the good parts of both parents and hence becomes even better. The reason that crossover does not work in our scenario is that a lot of information that can be used is stored in the event log, but this information can best be used with mutation operators to correct existing process trees. The information in the event log can be used to a lesser extent when trying to combine parts of mediocre process trees. Moreover the chance is small that two selected process trees each contain parts that together describe non-overlapping parts of behavior well. However, this is a requirement for crossover to combine two process trees into a better process tree. This leads us to believe that in our situation crossover is not beneficial for either the exploitation or the exploration aspect of the search. Although it is possible to enable crossover in our algorithm, we believe it does not improve search efficiency.

Our finding supports the observations in evolutionary algorithm research, known as the “crossover-mutation debate”. Some research [35, 83] suggests that crossover can be seen as a 'macromutation' operation, i.e., a combination of multiple mutation operations at once. Other research shows that some problems are better solved without crossover operations [167]. So, although genetic programming (which works on parse trees) heavily relies on crossover operations (and sometimes even disables mutation completely [35]), we have found that for the ETM*d* algorithm mutation is far more effective. For a more detailed discussion of the crossover-mutation debate we refer to the literature study of [163].

6.5 Candidate Selection

In the previous sections we discussed several ways to change candidates in the population. As is shown in Figure 6.1 and discussed in Section 4.4.2 candidates from the population should be selected to apply the changes on. This is another important phase for evolutionary algorithms. In the selection phase it is determined which candidates from the current generation are selected to survive into the next generation. Again, the balance between the exploration and exploitation aspects of the search is delicate and not straightforward. In current literature there is an ongoing discussion on which selection mechanism works



(a) Parent 1 (b) Parent 2

(c) Offspring 1 (d) Offspring 2

Figure 6.9: Example of crossover applied on two parents, creating two offspring.

best in particular situations [35, 100]. We have chosen sigma scaling because it maintains a relatively constant selection pressure during the run of the evolutionary algorithm. Sigma scaling scales the quality of a candidate according to the mean and standard deviation of the quality of the whole population. Equation 6.1 shows the exact formula that is applied. The value after scaling (q_σ) is defined as the original quality (q) minus the mean quality of the population (μ), divided by 2 times the standard deviation (σ) of the quality of the population.

$$q_\sigma = \max(1 + \frac{q-\mu}{2\sigma}, 0) \tag{6.1}$$

So, when $q_\sigma = 0$ this means that the candidate is worse than twice the standard deviation below average, and is not going to be selected. If $q_\sigma = 1$ this means that the candidate is exactly average. Candidates with higher values for q_σ therefore have a bigger chance of being selected. Effectively this means that extremely good candidates become less good, to reduce the probability that such candidates are selected. At the same time it reduces the probability that bad candidates are selected.

Sigma scaling scales the original quality values, to take the mean quality and standard deviation of the population into account. However, a selection or sampling method still has to be applied. We apply stochastic universal sampling [77, 100] (also see Section 4.4.2), because it is a fitness-proportionate selection strategy. Stochastic universal sampling assigns each candidate a chance of selection proportional to their quality, like roulette wheel selection does. However, stochastic universal sampling randomly chooses a point on the roulette wheel to start from, and then advances equally sized steps around the roulette wheel to select the number of candidates required. This ensures equal spread of the candidates selected with zero bias [34]. The main benefit is that this results in a diverse collection of selected candidates.

6.6 Termination Conditions

The only remaining element of the evolutionary algorithm to be defined is that of termination. The purpose of this element was discussed in more detail in Section 4.4.4. One of the most straightforward termination conditions is to stop when the perfect candidate has been found. However, in general it is not known whether such a candidate exists. And even if it exists, it is not guaranteed that the evolutionary algorithm will discover it in a reasonable time. Moreover, in our case the generalization metric as discussed in Section 5.6.1 cannot reach a perfect score of 1, so the candidate with a perfect quality actually does not exist.

Since it is not known whether the best possible candidate has been found, other aspects such as elapsed time or the number of attempts need to be considered. However, there is no guarantee that the best possible candidate is actually found by the evolutionary algorithm. Therefore termination conditions should balance the quality of the candidate and the elapsed time, using the probability of a better candidate being discovered in the near future.

In Section 7.1 we perform several experiments comparing two of the most commonly used termination conditions: by number of generations and the quality stagnation termination. As a result of these experiments we observe that for a practical setting the quality stagnation termination is a good termination condition. However, in this thesis we use the number of generations termination condition. By setting this to a relatively high number of generations, we can better compare different runs of the ETM*d* algorithm. Furthermore, in this thesis we mainly focus on the quality of the process models discovered.

In real-life scenarios we propose to combine the stagnation termination with a 'live' version of the ETM*d* algorithm. In the live version the current best result is shown directly to the user, allowing them to manually stop the execution if desired. Furthermore, while the ETM*d* algorithm is still running, the current best result can already be used to perform preliminary analysis, while the ETM*d* algorithm continues its search for better candidates.

6.7 Balancing Search Space Exploration and Exploitation

As discussed in Section 4.3, the *exploration* and *exploitation* aspects of the search by an evolutionary algorithm should be balanced. The population should be diverse to allow for exploration, but also contain several good candidates for exploitation. Moreover, it is important that the whole search space can be covered by a combination of the population creation, mutation and crossover operators used.

Table 6.1 shows an overview classifying all discussed operators. In general, exploration is mainly addressed by the random operators defined. The random operators combined allow for every possible process tree to be eventually created, when these operators are run infinitely many times. Thus, they clearly support exploration. Exploitation is addressed by the smarter guided operators

Table 6.1: Overview of the different operators defined and their contribution to the search aspects of exploration and exploitation.

Operation	Section	Exploration	Exploitation
Random Tree Creation	6.1.1	✓	
Trace-Model Creation	6.1.2		✓
Creation by other Algorithms	6.1.3		✓
Random Node Removal	6.2	✓	✓
Random Node Addition	6.2	✓	✓
Random Node Mutation	6.2	✓	✓
Normalization Mutation	6.2		✓
Remove Useless Nodes Mutation	6.2		✓
Replace Tree	6.2	✓	
Shuffle Nodes	6.2		✓
Guided Behavior Removal	6.3.1		✓
Guided Behavior Addition	6.3.2		✓
Guided Behavior Change	6.3.3		✓
Crossover	6.4	✓	
Candidate Selection	6.5	✓	✓

since they use knowledge to almost certainly improve the quality of a process tree and thus exploit the neighborhood of the process model.

The random mutation operators that add, remove and change nodes in the process tree can contribute to both the exploration and exploitation aspects of the search. If these operations are applied high in the tree the possible effect is large, which is more explorative, and when applied close to or on the leaves, the effect is smaller and thus more exploitative. The other random mutation operators of normalization, removal of useless nodes, and shuffling of nodes apply behavioral-equivalent mutation operations. This is an exploitative change which is mainly used to increase the diversity of the population, and to slightly change the tree so that other mutation operators can improve the model. The mutation operation of replacing a whole process tree using one of the tree creation techniques is of an explorative nature since it introduces new candidates to the population. Crossover is also an explorative operator since it combines parts of two trees to create two significantly different trees.

Candidate selection supports both the exploration and exploitation aspects of search. It has a high chance of selecting good candidates, supporting the exploitative aspect. However, by also allowing lower quality candidates to be selected, exploration is supported.

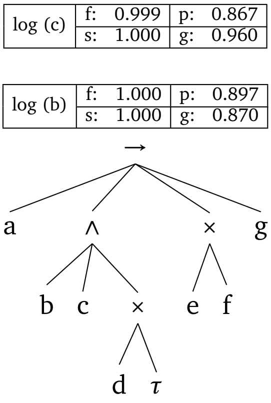
6.8 Application Using a Running Example

Now that all the ingredients of the ETM*d* algorithm have been discussed for the scenario of process discovery, we can apply ETM*d* on the running example used throughout this thesis (see Section 1.2). The example is illustrated in Figure 6.10. Figure 6.10a shows the process tree that can be considered the system that generated the behavior. From the process model two event logs are simulated. In Figure 6.10b the event log without exceptional behavior, generated from this process model, is shown. The event log contains 11 unique traces and 100 traces in total. Additionally an event log is generated that contains exceptional behavior, as shown in Figure 6.10c. This event log consists of 1,020 traces of which the first 1,000 traces are taken from the event log without exceptional behavior, which is replicated 10 times. The last 20 traces of the event log with exceptional behavior do not perfectly fit the process model of Figure 6.10a.

6.8.1 Searching for the Best Process Tree

The ETM*d* algorithm can search for the best process tree by weighing the quality dimensions. As shown in [49] the overall quality can best be calculated by assigning a weight to replay fitness of 10, precision 5, and both generalization and simplicity once. In the first experiment we run the ETM*d* algorithm on the event log without exceptional behavior. We search for the best tree by weighing replay fitness times 10, since it relates the modeled with the observed behavior. Precision is weighed 5 times since the model should not describe too much additional behavior. Simplicity is weighed once to prevent useless nodes. Generalization is given a weight of 0.1 since it is of less importance. The ETM*d* algorithm has been executed 30 times for 10,000 generations, and the result that is closest to the average of these runs is shown in Figure 6.11a. This is not exactly the same process tree as the (unknown) system that generated the behavior. The major difference is the fact that there is no option to skip activity `d`, which is only observed in 10 out of 100 traces. The resulting process tree scores a bit worse on replay fitness (0.995 instead of 1.000 for the 'system model'). It is however a more precise description of the observed behavior since precision of the discovered tree is 0.996 instead of 0.897 of the system model. Generalization also improved to 0.886 from 0.870. Simplicity is perfect for both trees since there are no useless nodes.

When the ETM*d* algorithm is run on the event log with exceptional behavior we obtain the process model shown in Figure 6.11b. The behavior of the discovered process tree allows for activity d to be skipped. Furthermore, it allows to execute only activity d and skip both activities b and c, which is observed in 4 traces. It is however not allowed to skip both activities e and f (observed in 7 traces) or execute both (observed in 2 traces). The discovered process model has the same score on replay fitness as the 'system model', but scores much better on precision (0.996 instead of 0.867). The increase in precision comes at the expense of generalization, which reduced from 0.960 to 0.930 due to the dupli-



(a) System that generated the event log, represented by a process tree.

(b) Generated event log of 100 traces, without exceptional behavior.

Trace	#
a b c d f g	38
a b d c f g	26
a b d c e g	12
a b c f g	8
a b c d e g	6
a d c b f g	4
a c d b f g	2
a c b e g	1
a d b c f g	1
a d b c e g	1
a c b f g	1

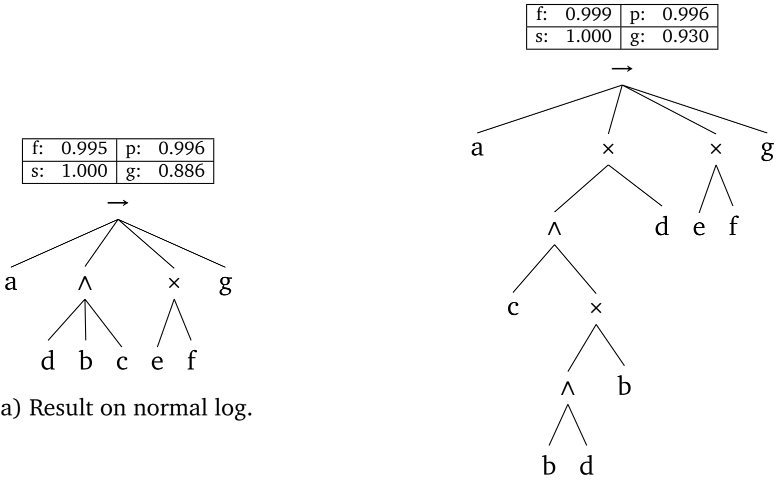
(c) Generated event log of 1,020 traces, with exceptional behavior.

Trace	#
a b c d f g	380
a b d c f g	260
a b d c e g	120
a b c f g	80
a b c d e g	60
a d c b f g	40
a c d b f g	20
a c b e g	10
a d b c f g	10
a d b c e g	10
a c b f g	10
a c b d g	4
a d e g	4
a b c g	3
a c f g	3
a b c d e f g	2
a b d e g	2
a c d f g	2

Figure 6.10: Running example used to discover a process tree (see Section 1.2).

cation of activity d. Simplicity is perfect for both trees since there are no useless nodes.

These experiments both show that a single result is not always sufficient. Questions like “what does the process tree with perfect replay fitness look like” and “at what cost can perfect replay fitness be achieved” are left unanswered. Therefore we apply the ETM*d* algorithm again on the event log with exceptional behavior, while constructing a Pareto front of solutions.



(a) Result on normal log.

(b) Result on event log with exceptional behavior

Figure 6.11: Results of the ETM*d* algorithm on the running example event logs.

6.8.2 Discovery of a Pareto Front

As discussed in Section 4.4.1, instead of aggregating the different quality values, a Pareto front can be constructed that keeps the different quality dimensions separated. Here we discuss some of the process trees contained in this Pareto front. A more detailed discussion of the characteristics of the Pareto front is provided in Section 7.1.3.

The Pareto front discovered by the ETM*d* contains 74 process trees, each with different trade-offs between the quality dimensions considered (see Figure 7.13). Two process trees from the Pareto front are shown in Figure 6.12. The process tree of Figure 6.12a is exactly the process tree as discovered by the ETM*d* algorithm with weighted quality dimensions on the running example without exceptional behavior. The process tree in the Pareto front with the best precision for a perfect replay fitness is shown in Figure 6.12b. This process tree explains all of the observed behavior, but not in a generalizing way.

The process tree as discovered by the ETM*d* algorithm using weighted quality dimensions is not present in the Pareto front. This is caused by the fact that the ETM*d* algorithm did not consider this process tree (yet), because it would be included in the Pareto front, had it been discovered. The discovered Pareto front is discussed in more detail in Section 7.1.3.

6.9 Results of Existing Process Discovery Algorithms

In this section we apply several process discovery algorithms on the two running examples also used to evaluate the ETM*d* algorithm in Section 6.8. Some algorithms resulted in *unsound* process models, i.e., process models that are not semantically correct (see Section 3.1.1). We translated the behavior of each proces model to a process tree, in order to measure the quality of the result. Where applicable, we stayed as close as possible to the intended behavior of

In Section 6.9 we revisit results presented in [49, 52]:

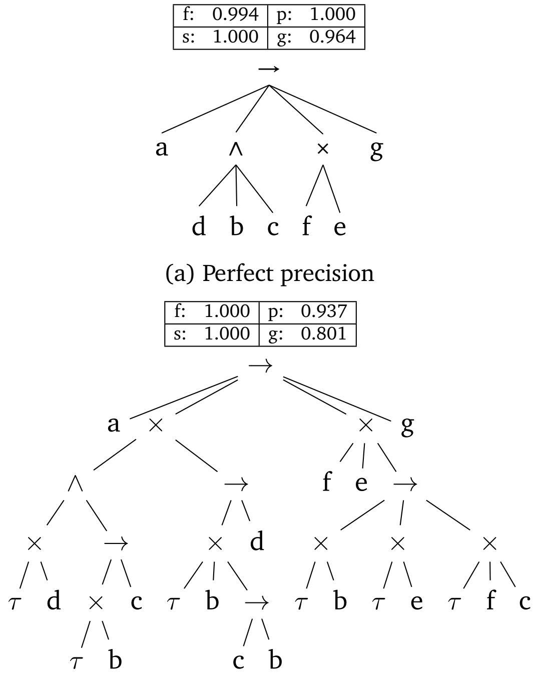
- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In R. Meersman, H. Panetto, T.S. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I.F. Cruz, editors, *OTM Conferences (1)*, volume 7565 of *Lecture Notes in Computer Science*, pages 305–322. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33605-8
- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, 2014

.

the original model. Each process tree is then evaluated on the four quality dimensions of process discovery, i.e., replay fitness, precision, generalization and simplicity.

For all algorithms the publicly available version, as included in the ProM process mining framework version 6.3, has been used, unless mentioned otherwise.

In the remainder of this section we discuss the results and most common issues of these algorithms.



(a) Perfect precision

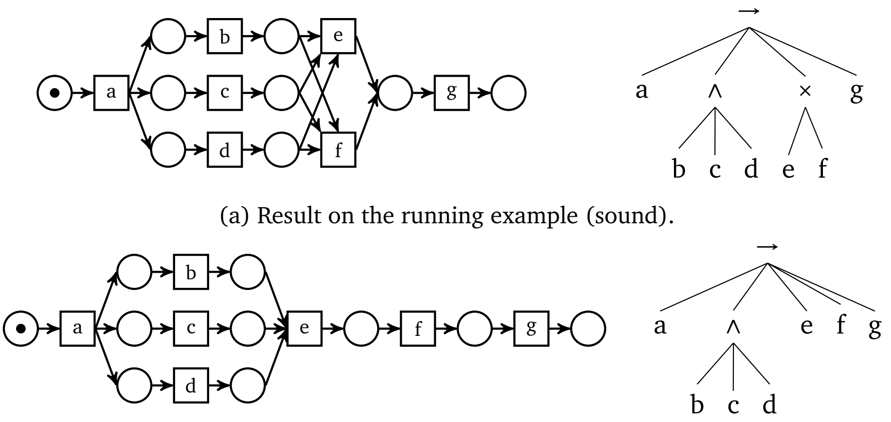
(b) Perfect Replay fitness.

Figure 6.12: Examples of process trees with slightly different trade-offs between the quality dimensions.

6.9.1 The α-Algorithm

One of the first process discovery algorithms is the α-algorithm [20]. The α-algorithm is one of the most basic process discovery algorithms, and does not take any parameters. The result of running the α-algorithm on our running example is the Petri net shown at the left hand side of Figure 6.13a. This Petri net can be easily translated to the process tree notation without changing the intended behavior. The process tree that is the result of this translation is shown at the right hand side of Figure 6.13a. The activities `e` and `f` in the Petri net also function as a silent parallel join. In the process tree this has been split in a control flow node (as parents of `b`, `c` and `d`) and a choice between activities `e` and `f`.

Applying the α-algorithm on the running example event log with exceptions, results in the process model and process tree translation as shown in Figure 6.13b. The only difference with the previous result is the fact that there no longer is a choice between activities `e` and `f`, but that both need to be executed. This also shows the main disadvantage of the α-algorithm: it is not resilient to exceptional behavior. Only two exceptional traces introduced the activity dependencies which removed the choice between activities `e` and `f`, reducing replay fitness. For real life event logs the α-algorithm often does not result in usable process models because of its sensitivity to exceptional behav-



(a) Result on the running example (sound).

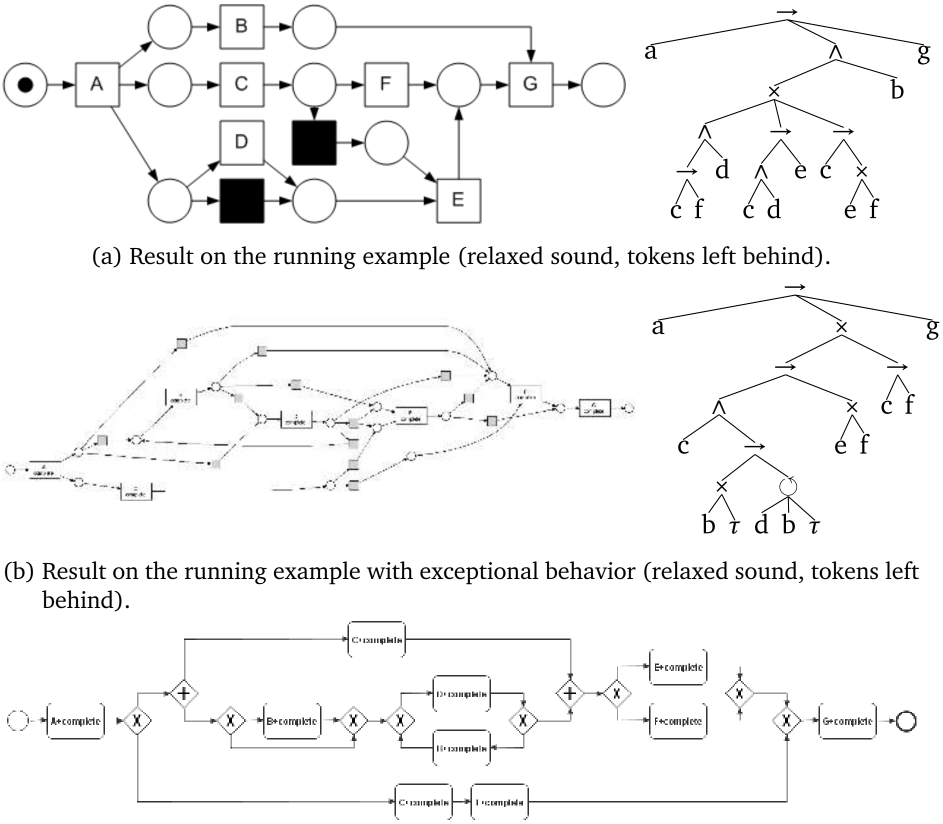
(b) Result on the running example with exceptional behavior (coincidentally sound).

Figure 6.13: Results of the α-algorithm [20] on the running examples.

ior. Furthermore, exceptions quickly result in unsound models being produced by the α-algorithm.

6.9.2 Genetic Miner

The Genetic Miner [29] is run with a population of 20, a target fitness of 1.0 and for a maximum of 10,000 generations. The result is a relaxed sound Petri net since tokens are left before task e when executing activities $\langle a, b, c, d, f, g \rangle$.



(a) Result on the running example (relaxed sound, tokens left behind).

(b) Result on the running example with exceptional behavior (relaxed sound, tokens left behind).

(c) BPMN translation of the process tree of Figure 6.14b.

Figure 6.14: Result of the genetic miner [29] (Relaxed sound, tokens left behind).

When translating the behavior described by this model, for example the fact that the activities b and d are parallel to f, we obtained the process tree as shown at the right hand side in Figure 6.14a. Although the translated process tree can replay all observed behavior, the process tree is not very precise.

The result on the event log with exceptional behavior shows a similar process tree as a result, as shown in Figure 6.14b. However, the resulting Petri net is hard to read, which is caused by the many silent transitions moving tokens. The process tree can replay most of the observed behavior but not in a very precise way.

The Genetic Miner focusses mainly on replay fitness, which results in process models that are less precise. Generalization is not considered, since its interpretation in the Genetic Miner is the inverse of precision (see Section 5.6.2). The result on the running example with exceptional behavior also shows that the resulting process model is not easy to understand. Although the process tree looks complicated, Figure 6.14c shows the BPMN translation of the process tree, which is easier to understand.

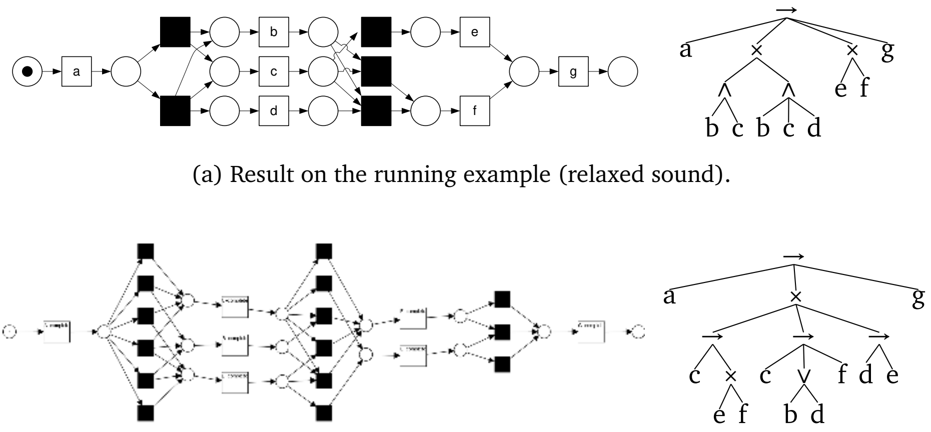
6.9.3 Heuristic Miner

The heuristics miner [180] has been developed to be more resistant to exceptional behavior than most other process discovery algorithms. When applied on the running example the Petri net as shown on the left hand side of Figure 6.15a is obtained, after converting the Heuristics net to a Petri net. All thresholds and other default settings (i.e., ‘all tasks connected’ enabled and ‘long distance dependency’ disabled) were maintained. Other settings were experimented with but did not result in a process model with a better quality score. Unfortunately, the resulting Petri net is not sound but relaxed sound. Tokens are left in the Petri net for instance when the following firing sequence is executed: $\langle a, c, e, g\rangle$. Tokens remain before or after activity b. Furthermore, if the bottom left silent transition is taken, tokens also remain before or after activity d.

Since the Petri net is not sound, it is not possible to directly translate it to a process tree. Therefore the process tree as shown in Figure 6.15a represents a sound interpretation of the intended behavior of the Petri net. Most notable is the choice between two different parallel branches, one with d and the other without. This is how the process model was meant as discovered by the heuristic miner, which is indicated by the two silent transitions in the beginning of the Petri net. The process tree can correctly replay all behavior of the event log.

The resulting Petri net for the event log with exceptions is shown in Figure 6.15b. There are a lot of silent transitions that allow for different combinations of activities being executed. This Petri net is again relaxed sound, since after the execution of the trace $\langle a, b, c \rangle$, tokens are always left in the place after d. The Petri net is also not able to replay the behavior, as can be seen from the process tree translation.

The results of the heuristics miner are in general relaxed sound, which for most analysis techniques is not sufficient. And although the heuristics miner is able to handle exceptional behavior, it results in low replay fitness scores in case the behavior is slightly more complicated. The heuristics miner however does consider precision since the behavior of the process model is restricted. However, this comes at the cost of generalization. Finally, the resulting process models are not easy to interpret since the different relationships between activities are encoded with separate transitions.



(a) Result on the running example (relaxed sound).

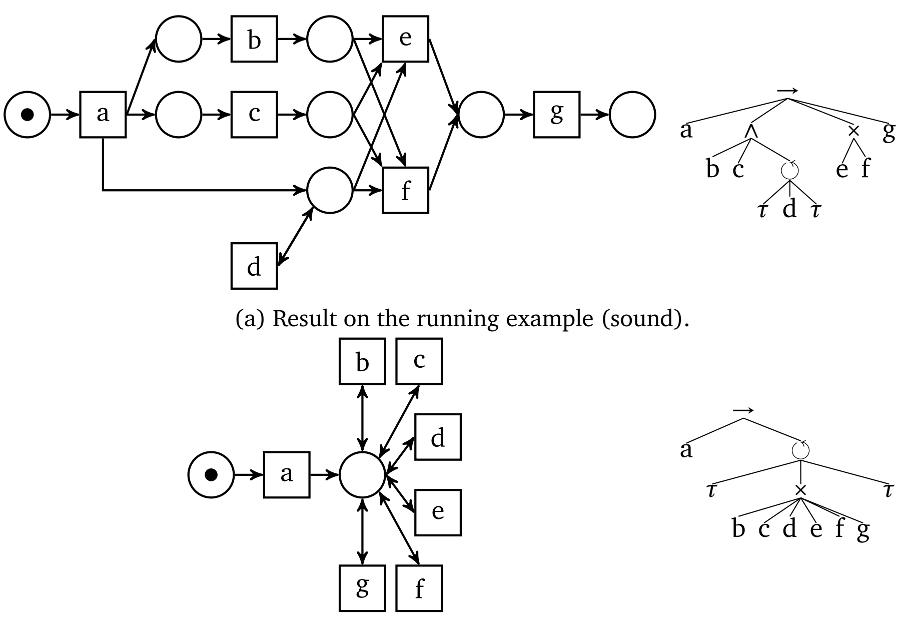
(b) Result on the running example with exceptional behavior (relaxed sound).

Figure 6.15: Result of the heuristic miner [180] (relaxed sound since tokens are left behind).

6.9.4 The ILP Miner

A process discovery algorithm that ensures perfect replay fitness is the ILP miner [182]. The result of running this algorithm on the running example's event log is shown at the left hand side in Figure 6.16a. This result is obtained by using the following, mostly default, parameters: 'Java-ILP and LpSolve' as solvers, 'Petri net (empty after completion)' as ILP variant, 'number of places' set to 'per causal dependency', and the option 'Search for separate initial places' is enabled. None of the ILP extensions were enabled. The ILP miner produced the Petri net as shown in Figure 6.16a.

This Petri net can be directly translated to a process tree, without changing its behavior. The guarantee of the ILP miner, that it always produces a perfectly fitting process model, still holds for the process model produced. However, this comes as the cost of precision since the model allows for loops of activity d while the event log never contains more than one instance of d per trace.



(a) Result on the running example (sound).

(b) Result on the running example with exceptional behavior (relaxed sound).

Figure 6.16: Result of the ILP miner [182] (Ensuring empty net after completion).

Applying the ILP miner on the event log with exceptional behavior demonstrates this even stronger. The resulting process model is able to express all observed behavior. However, except for activity a, all activities are in a so called 'flower' construct. This construct consists of a place, from which one or more activities take a token and put it back. The result, in this case, is that all activities, except activity a, can be executed in any order, and zero or more times, without restriction. Hence, this construct does not add knowledge about the observed behavior, since in essence it allows for all possible behavior. Furthermore, this process model is relaxed sound, because when the final marking is reached (the place in the flower construct), some transitions are still enabled.

The ILP miner often results in relaxed sound models, but with a perfect replay fitness. This however comes at a significant cost to precision, as is shown by the result on the running example with exceptional behavior. For larger real-life event logs the resulting process models are not simple since they contain many relationships, or many unconnected transitions (which is not a sound process model). Using other settings than used here may result in a more precise model with many more places.

6.9.5 Inductive Miner

The Inductive Miner [121] directly discovers process trees from an event log. We have used the version in ProM 6.4, where we chose the IMin variant, which is able to deal with infrequent behavior. The result on the running example event log is shown in Figure 6.17a. It captures most of the behavior correctly, but places activity d in a flower construct, similar to the ILP miner. So although this process tree can replay all observed behavior, it is not very precise.

Application of the Inductive Miner on the running example with exceptional behavior results in the process tree as shown in Figure 6.17b. Here we see that all activities except a and g are placed in flower constructs. Just like the ILP miner this results in a process model that is able to replay all observed behavior, but does not describe the event log in a precise way.

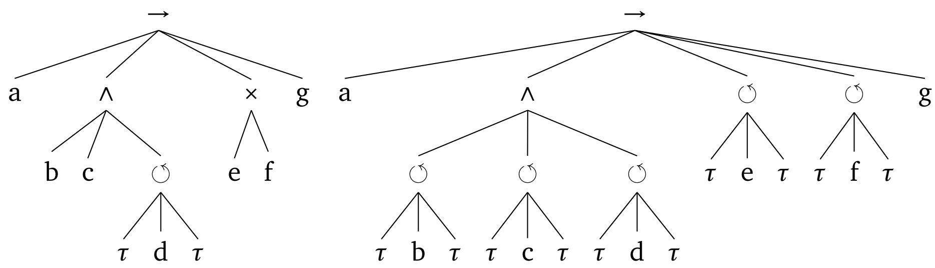
The Inductive Miner is one of few process discovery algorithms that also work on process trees. This results in simpler process models, especially when translated to the preferred modeling notation of the user. However, the Inductive Miner takes a constructive approach, where more emphasis is put on replay fitness than on precision. This is shown by the imprecise process model for the running example event log with exceptional behavior. Although it should be noted that there are parameter settings where a better process model is produced, requiring a user to experiment with parameter settings to obtain the best process model is not desirable.

6.9.6 Language-based Region Theory

The result of language-based region theory [37] can be obtained by running the ILP miner plug-in and setting the number of places to 'Basic Representation', disabling the 'search for separate initial places' checkbox and checking the option to discover an 'Elementary Net'. This produces the Petri net that is shown in Figure 6.18a. It is clear to see that the resulting model is overly complex and incomprehensible.

The translation to a process tree results in the process tree shown in Figure 6.18a. Since there is no option to skip d the replay fitness score is not perfect. The process model also includes activities e, f and g in the parallel part, which results in a lower precision, since these activities are always strictly executed at the end of the process. Simplicity of the process tree is perfect since each activity occurs exactly once. Note however that the translation from the Petri net to the process tree simplified the model drastically, while maintaining its behavior. This also indicates the main disadvantage of this technique: it results in incomprehensible process models that can be simplified. The question is however if this simplification can be performed automatically.

This is also demonstrated when this technique is applied on the event log with exceptions. The resulting Petri net is almost impossible to understand. The process tree shown next to the Petri net in Figure 6.18b describes the intended



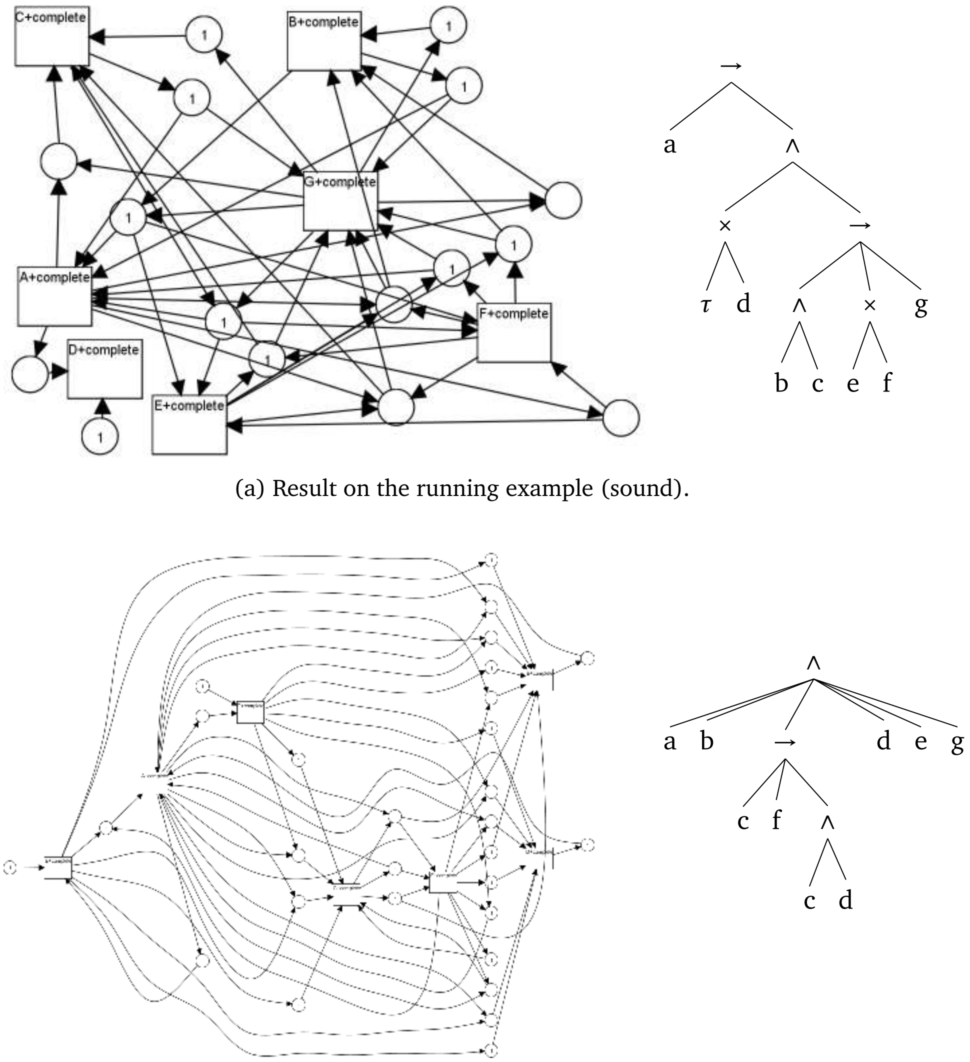
(a) Result on the running example (sound).

(b) Result on the running example with exceptional behavior (sound).

Figure 6.17: Results of the Inductive Miner.

behavior of the Petri net. The model has several issues when replaying the observed behavior. For instance, both `e` and `f` have to be executed and activities `c` and `d` have to be executed twice according to the model. Hence the replay fitness is bad, and precision is not high either. But most importantly, the Petri net that is the result of the language-based region theory is very hard to interpret.

Language based region theory has as main disadvantage that the resulting class of Petri net, an elementary net, is hard to interpret. Moreover, translating an elementary net to simpler notations is in general not easy. The resulting process models also score low on replay fitness, precision and generalization.



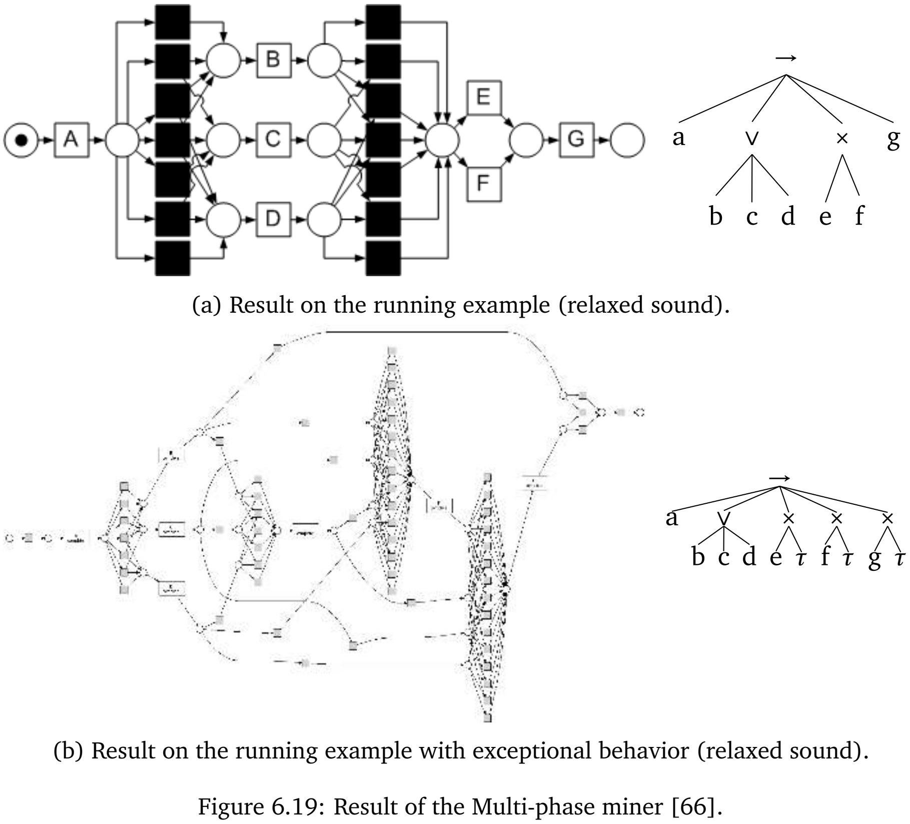
(a) Result on the running example (sound).

(b) Result on the running example with exceptional behavior (sound).

Figure 6.18: Result of the language-based region theory [37] (The models are overly complex and incomprehensible, but sound).

6.9.7 Multi-phase Miner

The result after running the Multi-phase miner [66] as included in ProM 5.2, with the default settings, results in an EPC model. Converting this EPC to a Petri net results in the Petri net as shown at the left hand side of Figure 6.19a. The process model is relaxed sound but is not easy to understand due to all the silent transitions before and after activities b, c and d. The process tree relations show that all these three activities are included in an ∨ construct, and can therefore be skipped. Although this process model can replay all observed behavior, it is not very precise since activities b and c are never skipped in the event log.



(a) Result on the running example (relaxed sound).

(b) Result on the running example with exceptional behavior (relaxed sound).

Figure 6.19: Result of the Multi-phase miner [66].

The result of the multi-phase miner on the event log with exceptions results in a sound but incomprehensible Petri net as shown in Figure 6.19b. The original EPC model is easier to understand but due to the combination of ×-splits and ∨-joins still hard to read. The intended behavior, as is described by the process tree shown in Figure 6.19b, however, shows that the actual behavior is not very complicated. The resulting process model is able to replay all observed behavior, including the exceptional traces. However, precision is not very high since the model allows for a lot of different behavior.

The multi-phase miner guarantees relaxed sound results, but not sound results. It furthermore guarantees to provide a result with perfect replay fitness and the highest possible precision it can obtain. However, more precise process models exist than the ones discovered by the multi-phase miner. Moreover, the resulting process models are not easy to interpret, mainly because of the many silent transitions in the Petri net.

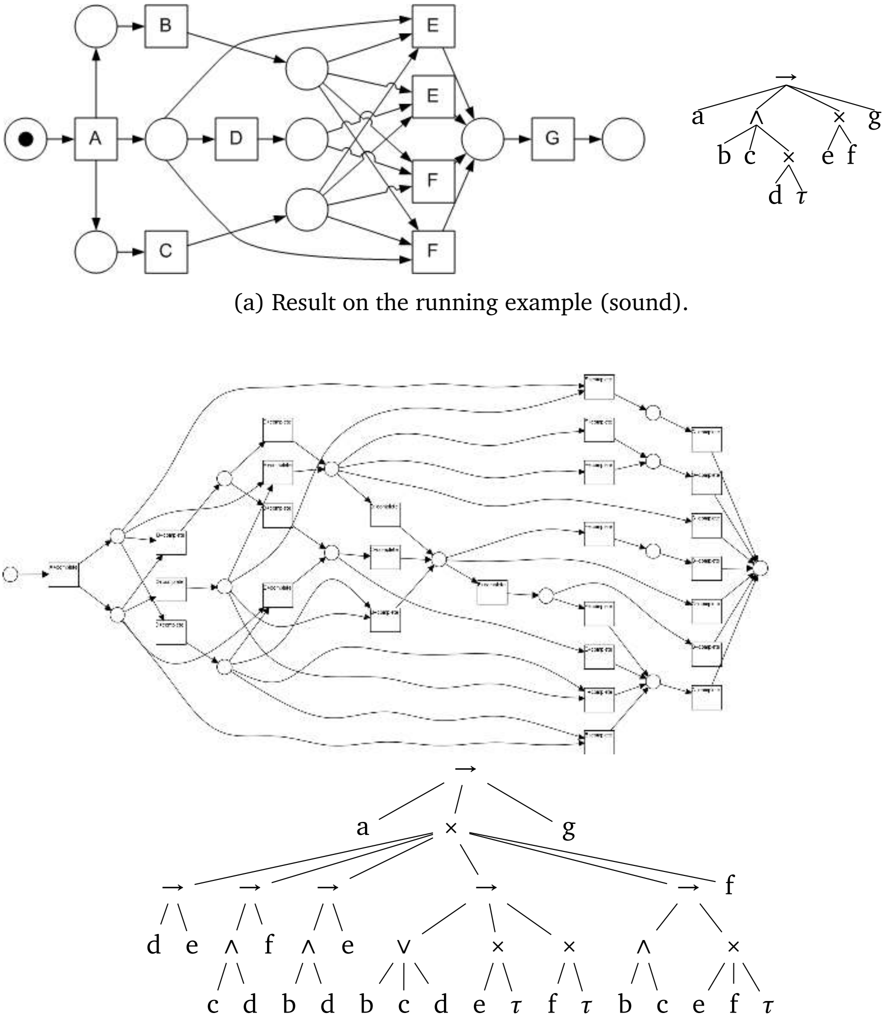
6.9.8 State-based Region Theory

Applying state-based region theory [31, 56, 76, 164], by executing the plug-in 'Mine Transition System' followed by a conversion to Petri nets using region theory, results in the Petri net as shown in Figure 6.20a. For the transition system mining, the default settings are used with unlimited maximum set size and inclusion of all activities.

The resulting Petri net is sound and includes the option to execute activities `e` and `f` without executing `d` explicitly, by duplicating activities `e` and `f`. We have translated this Petri net to a process tree without duplicating activities `e` and `f`, which would have reduced simplicity and generalization. The models can replay all observed behavior. However, the original Petri net is not very easy to understand, mainly due to the duplication of activities `e` and `f`.

The result on the event log with exceptional behavior is again a rather large Petri net, as shown in Figure 6.20b. The process tree translation is also rather complex due to activity duplication. Although the process model can replay all observed behavior, it is not very precise.

In general state-based region theory results in sound process models with a guaranteed perfect replay fitness. However, generalization is not considered, since irregular behavior is specifically captured to increase precision. A clear example of this is shown in the result for the running example with exceptional behavior. This however also results in a Petri net that is difficult to understand.



(a) Result on the running example (sound).

(b) Result on the running example with exceptional behavior (sound).

Figure 6.20: Result of the state-based region theory.

Table 6.2: Quality of results of process discovery algorithms on the running example event log. *Italic* algorithm names indicate unsound process models. Best models are indicated in **bold**.

Algorithm	Fig.	Overall	f	p	s	g
ETM*d*	6.11a	0.9951	0.9952	**0.9960**	**1.0000**	0.8859
α-algorithm	6.13a	0.9951	0.9952	**0.9960**	**1.0000**	0.8859
Genetic Miner	6.14a	0.9675	**1.0000**	0.8996	**1.0000**	0.7799
Heuristic Miner	6.15a	**0.9954**	**1.0000**	0.9884	**1.0000**	0.8424
ILP Miner	6.16a	0.9556	**1.0000**	0.8593	**1.0000**	**0.8913**
Inductive Miner	6.17a	0.9556	**1.0000**	0.8593	**1.0000**	**0.8913**
Language-based region theory	6.18a	0.9456	**1.0000**	0.8111	**1.0000**	0.8745
Multi-phase Miner	6.19a	0.9488	**1.0000**	0.8373	**1.0000**	0.8859
State-based Region Theory	6.20a	0.9471	**1.0000**	0.8966	**1.0000**	0.8702

6.9.9 Why Existing Algorithms Fail

The results of existing process discovery algorithms as discussed in this section are summarized in Table 6.2 and Table 6.3. The process models and the quality metrics clearly indicate that, even on our small artificial dataset, existing algorithms have difficulties balancing the quality dimensions. Existing algorithms had issues especially on the running example with exceptional behavior. The algorithms that guarantee perfect replay fitness (i.e., the ILP miner, Inductive Miner, Multi-phase Miner and State-based Region Theory), were only able to do so with a great reduction in precision. The other algorithms tried to balance the quality dimensions more but did not find a proper balance. Moreover, several algorithms produced an unsound result. The α-algorithm, which performed the best of all algorithms on the running example without exceptional behavior, was not able to discover an appropriate model for the event log with exceptional behavior.

Moreover, all process discovery algorithms make several notable assumptions. Most algorithms assume there is no exceptional behavior and thus try to describe all observed behavior. Hence, they focus only on the replay fitness quality dimension, and try to achieve perfect replay fitness. However, a more precise description, which often is also simpler, that still describes enough observed behavior can be often be found by relaxing the perfect replay fitness

Table 6.3: Quality of results of process discovery algorithms on the running example event log. *Italic* algorithm names indicate unsound process models. Best models are indicated in **bold**.

Algorithm	**Fig.**	**Overall**	**f**	**p**	**s**	**g**
ETM*d*	6.11b	**0.9976**	0.9990	0.9958	**1.0000**	0.9297
α-algorithm	6.13b	0.9676	0.9482	**1.0000**	**1.0000**	0.9638
Genetic Miner	6.14b	0.9547	0.9895	0.8775	**1.0000**	0.8810
Heuristic Miner	6.15b	0.8953	0.8322	**1.0000**	**1.0000**	0.9265
ILP Miner	6.16b	0.8151	**1.0000**	0.4054	**1.0000**	0.9677
Inductive Miner	6.17b	0.8921	**1.0000**	0.6532	**1.0000**	**0.9684**
Language-based region theory	6.18b	0.7774	0.8682	0.5374	0.9231	0.8811
Multi-phase Miner	6.19b	0.9279	**1.0000**	0.7699	**1.0000**	0.8984
State-based Region Theory	6.20b	0.9267	**1.0000**	0.7710	**1.0000**	0.6459

requirement.

Most process discovery algorithms have several options that can be changed, however the effects of these options are not always clear to the user. The optimal parameter settings are hard to obtain, especially in cases where the characteristics of the observed behavior are unknown, e.g. the number of exceptions.

6.10 Conclusion

In this chapter we have discussed all elements of the ETM*d* algorithm which is an implementation of the ETM framework tailored towards classical process discovery. We have discussed several ways to create the initial population in Section 6.1, from completely random to somewhat smart, including usage of existing process discovery algorithms. We also presented several mutation operators, some of which are random (Section 6.2) while others (Section 6.3) use information from the event log and the calculated alignments to apply more targeted changes to the process models. Additionally we discussed the crossover change operator in Section 6.4, though this operator does not improve the quality of the process models. Another aspect in evolutionary algorithms is selecting the candidates that survive to the next generation, which we discussed in Section 6.5. Deciding when to terminate the execution of the ETM*d* algorithm was discussed in Section 6.6. In all these phases of an evolutionary algorithm the exploration and exploitation aspects of the search should be considered. We discussed in Section 6.7 how all phases of the ETM*d* algorithm contribute to these aspects. The ETM*d* algorithm has been applied to two running examples in Section 6.8.1, and in Section 6.8.2 a Pareto front was constructed for these data sets. The results of the ETM*d* algorithm have been compared with that of existing process discovery algorithms in Section 6.9.

In the next chapter we apply the ETM*d* algorithm on more data sets, both artificial and from real life, and investigate the characteristics of the results in more detail.

Chapter 7

Application of Process Tree Discovery

In Section 6.8 we applied the ETM*d* on the running example event logs, and in Section 6.9 we compared the results with those of existing process discovery algorithms. In this chapter we evaluate three aspects of the ETM*d* algorithm and apply the ETM*d* algorithm on six real life event logs. Table 7.1 shows an overview of the different characteristics of the event logs used in this chapter.

The first aspect of the ETM*d* algorithm we evaluate is how it behaves when run for a longer period of time by investigating the quality of the results discovered per generation in Section 7.1. Next in Section 7.2 experiments are performed with different settings for guided versus random mutation. In Section 7.3 and Section 7.4 we apply the ETM*d* algorithm on real life data sets obtained in the CoSeLoG project and analyze the resulting Pareto front of process trees. Finally, in Section 7.5 we discuss the performance aspect of the ETM*d* algorithm for the experiments run in this chapter. Section 7.6 concludes this chapter.

7.1 Performance in the Limit

The ETM*d* algorithm was run 30 times on each of the two running examples depicted in Figure 7.1 for 10,000 generations, in order to investigate the long-term behavior.

Table 7.1: Statistics of the event logs used in this chapter.

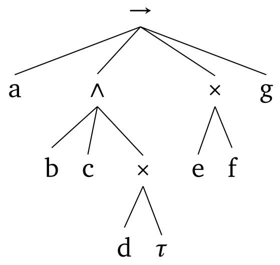
Log Name	#Traces	#Events	#Activities
Running example	100	590	7
Running example with exceptional behavior	1,020	5,994	7
Building Permits - Receipt Phase [46]	1,434	8,577	27
WABO1_BB [45]	54	131	15
WABO2_BB [45]	302	586	13
WABO3_BB [45]	37	73	9
WABO4_BB [45]	340	507	9
WABO5_BB [45]	481	845	23

7.1.1 Running Example without Exceptional Behavior

Figure 7.2 shows the overall quality of the best candidate discovered on the running example without exceptional behavior. The dashed line indicates the overall quality of the best candidates discovered, averaged over the 30 runs, which increased rapidly in the first generations. The average overall quality in the first generation was already 0.9557, and increased to an average of 0.9954 in the final generation. The continuous line, above the average line, shows the maximal overall quality over all runs. The dotted line shows the worst overall quality. Between runs there is some difference, but the average is close to the best overall quality. This indicates that, although there are differences between runs of the ETM*d* algorithm, the ETM*d* algorithm is able to discover good quality process models in all runs.

Figure 7.3 shows the average, maximum, and minimum overall quality over the first 100 generations. The average overall quality at generation 100 is 0.9946. This is only 0.0008 below the average overall quality after 10,000 generations. This shows that the ETM*d* algorithm quickly discovers reasonably good process models in early generations. Improving on these process models however requires more generations, with little impact on the overall quality.

Figure 7.4 shows the process trees closest to the average overall quality for that generation. For example, the average overall quality for generation 0 is 0.9557. The process tree found by one of the 30 runs in generation 0 with an overall quality closest to this value is shown in Figure 7.4a and has an overall quality of 0.9559. The process tree generated in generation 0 is clearly generated by the advanced tree creation discussed in Section 6.1.2. It allows for a particular sequence of activities b, c and d, which is the most frequent sequence in the event log. Moreover activity d can be skipped and activities e and f are in an exclusive choice. This results in perfect precision, but the replay fitness score can be improved. The ETM*d* algorithm achieves this already in generation 1 (see Figure 7.4b), where replay fitness improves slightly, at the cost of generalization, which is caused by the duplication of activity d. In generation 2 (see Figure 7.4c) the best model found on average has a further improved



(a) Process model that generated the event log

(b) Running example event log.

Trace	#
a b c d f g	38
a b d c f g	26
a b d c e g	12
a b c f g	8
a b c d e g	6
a d c b f g	4
a c d b f g	2
a c b e g	1
a d b c f g	1
a d b c e g	1
a c b f g	1

(c) Running example event log with exceptional behavior.

Trace	#
a b c d f g	380
a b d c f g	260
a b d c e g	120
a b c f g	80
a b c d e g	60
a d c b f g	40
a c d b f g	20
a c b e g	10
a d b c f g	10
a d b c e g	10
a c b f g	10
a c b d g	4
a d e g	4
a b c g	3
a c f g	3
a b c d e f g	2
a b d e g	2
a c d f g	2

Figure 7.1: Running example used to discover a process tree (see Section 1.2). In this chapter various experiments are conducted to see under which circumstances the ETM*d* algorithm is able to return a desired model.

replay fitness at the cost of simplicity (because of the →-operator under another →-operator). Generalization is also reduced because the tree size increased. Inspecting the result at generation 10 (see Figure 7.4d) we see that replay fitness is further improved, as well as simplicity and generalization. Replay fitness and generalization are further improved for the best candidate in generation 50 (see Figure 7.4e), but now at the cost of precision. The result at generation 100 (see Figure 7.4f) has a perfect score for replay fitness (i.e., 1.000), mainly at the cost

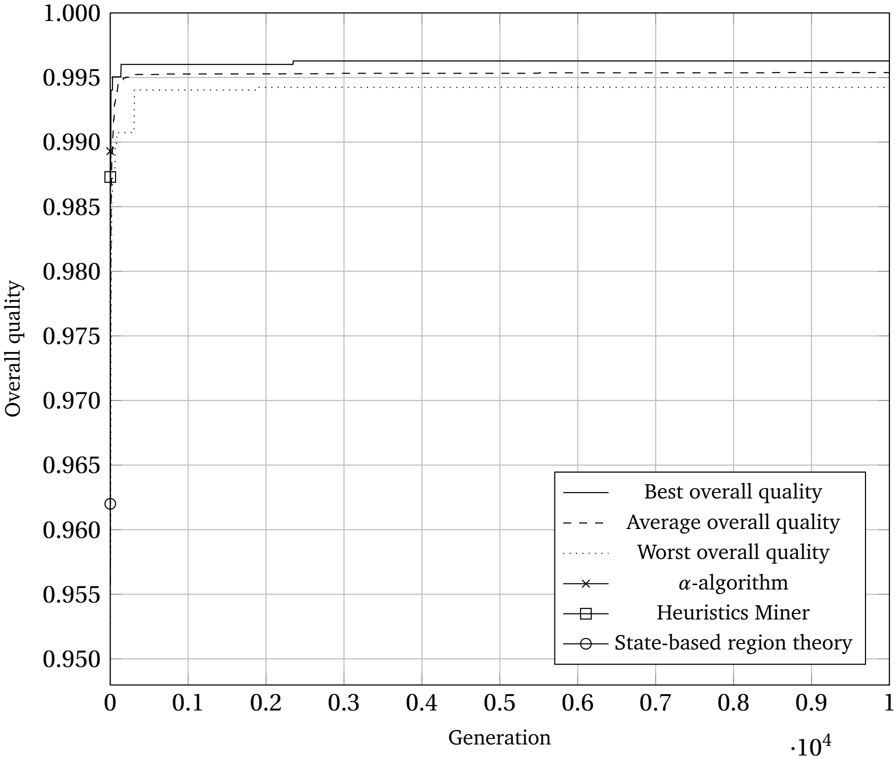


Figure 7.2: Minimum, average and maximum overall quality of the best candidates over 30 runs of ETM*d* on the running example, all 10,000 generations. This graph shows that the discovered process models quickly reach a high quality. Furthermore, the quality between different runs of the ETM*d* algorithm is similar.

of precision. This is caused by the difference in weights for the quality dimensions. Since replay fitness is weighted twice as much as precision (a weight of 10 versus 5), improving replay fitness at the cost of precision can still improve the overall quality of the candidate. The final result of the ETM*d* algorithm on this event log is shown in Figure 7.5a. This process tree has a higher precision, at the cost of replay fitness, than the process tree found in generation 100.

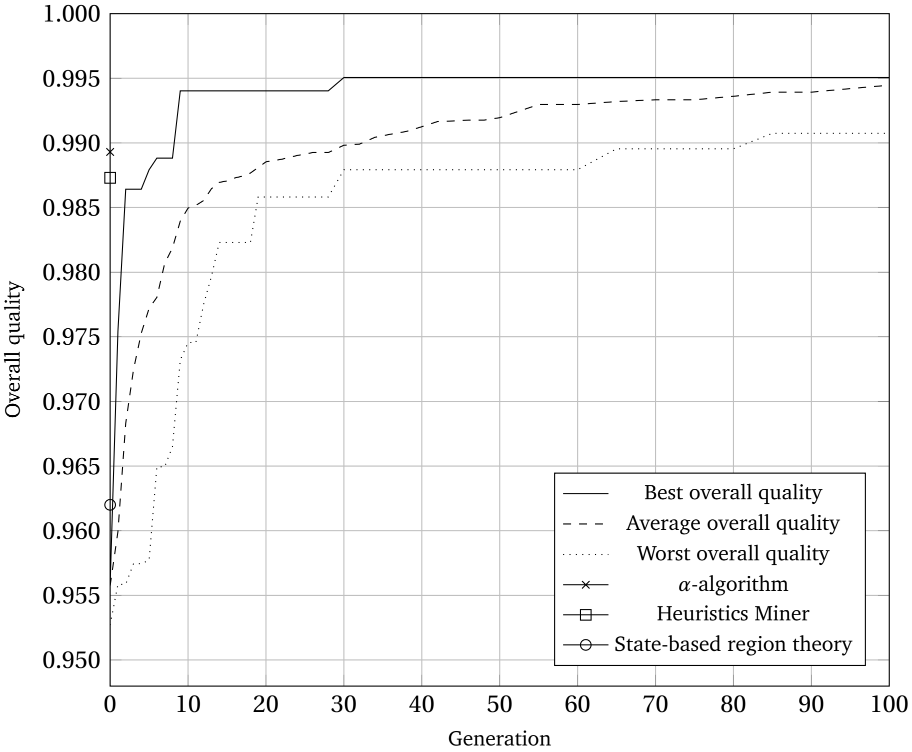
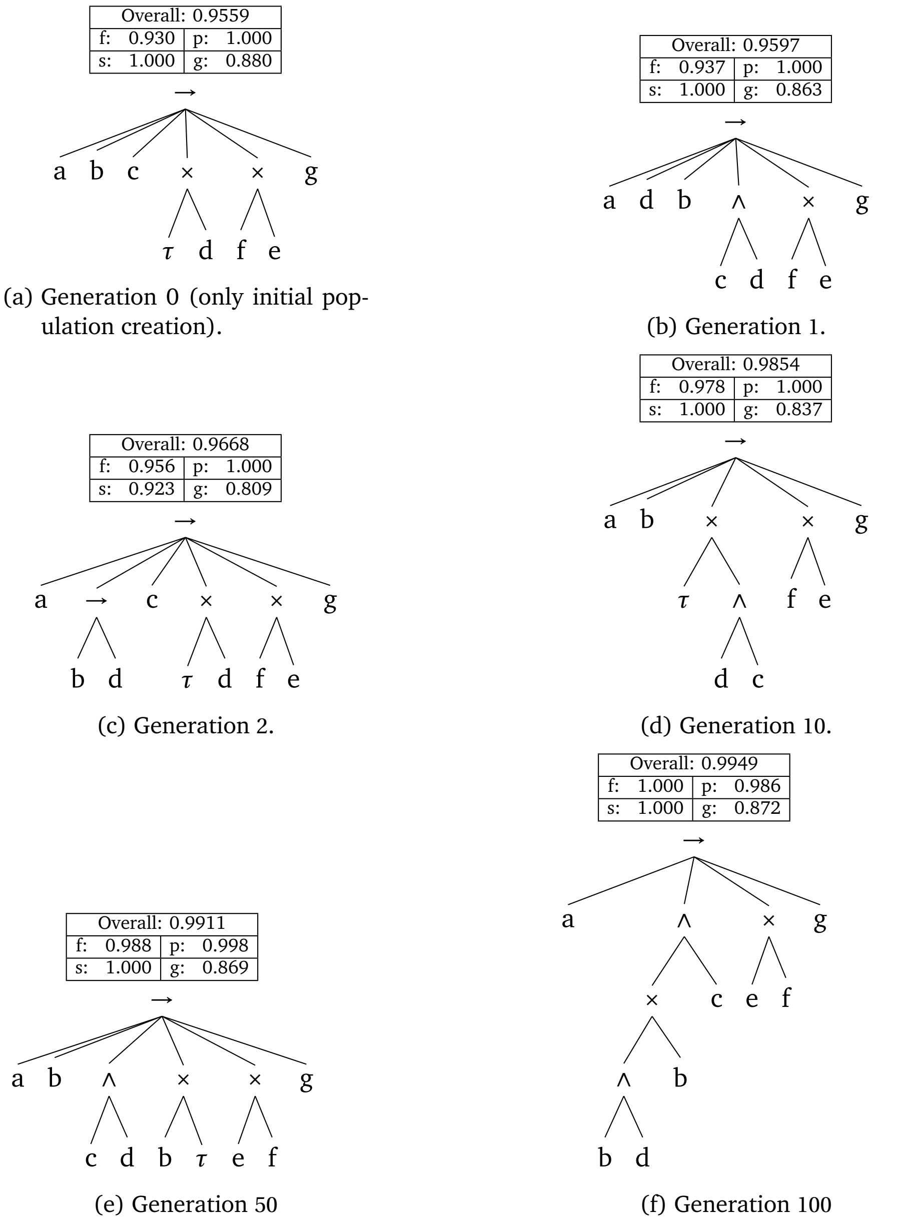


Figure 7.3: Minimum, average and maximum overall quality of the best candidates over 30 runs of the ETM*d* algorithm on the running example, first 100 generations. After 100 generations stable results outperform the α-algorithm, heuristics miner and state-based region theory results.



(a) Generation 0 (only initial population creation).

(b) Generation 1.

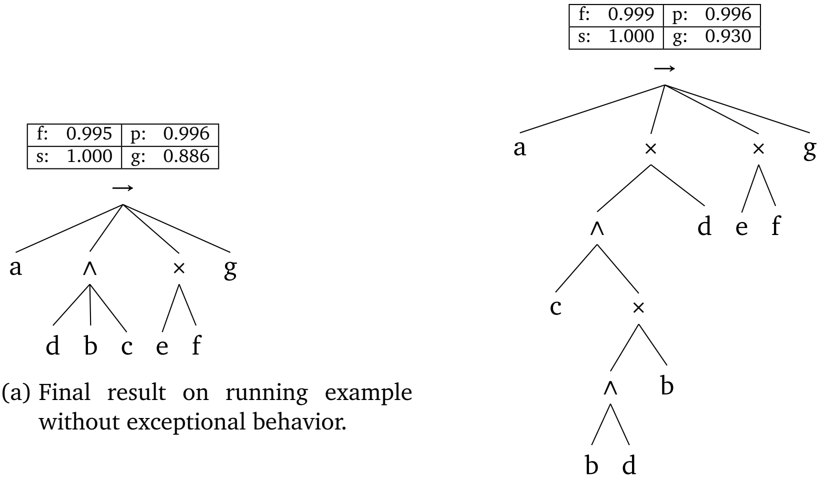
(c) Generation 2.

(d) Generation 10.

(e) Generation 50

(f) Generation 100

Figure 7.4: Process trees discovered by the ETM*d* algorithm on the running example in different generations. The process trees shown are closest to the average overall quality of all runs.



(a) Final result on running example without exceptional behavior.

(b) Final result on running example with exceptional behavior.

Figure 7.5: Results of the ETM*d* on the running examples. Both process trees balance the four quality dimensions.

7.1.2 Running Example with Exceptional Behavior

The graph in Figure 7.6 shows the minimum, average and maximum overall quality over 30 runs for the ETM*d* applied on the running example with exceptional behavior. The average overall quality in the first generation is again 0.9557, better than any of the other process discovery algorithms (see Table 6.3). The overall quality increased to an average of 0.9977 in the final generation. This is higher than for the running example without exceptional behavior be-

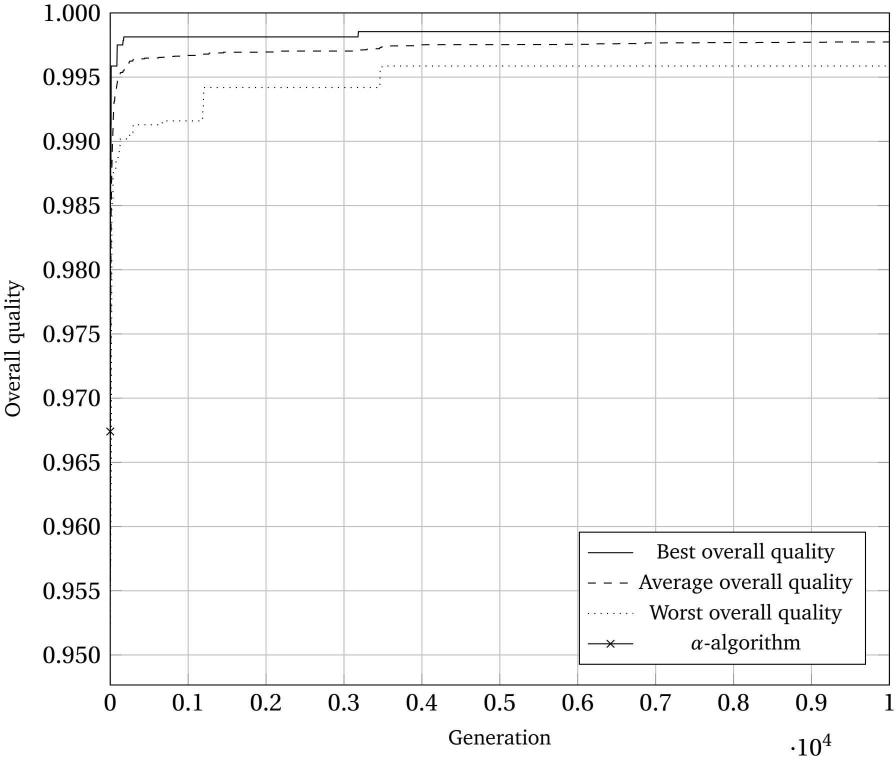


Figure 7.6: Minimum, average and maximum overall quality of the best candidates over 30 runs of ETM*d* on the running example with exceptional behavior, all 10,000 generations. The discovered candidates outperform the α-algorithm (and all other process discovery algorithms, which have an overall quality below 0.95) in early generations and converge quickly.

cause this event log contains more behavior resulting in a higher generalization score.

Figure 7.7 shows the overall quality development in the first 100 generations. The difference between the different runs is bigger than for the running example without exceptional behavior. The introduced exceptional behavior makes the observed behavior harder to capture in a process model, therefore the search has become more difficult. However, after generation 10 a better process model was

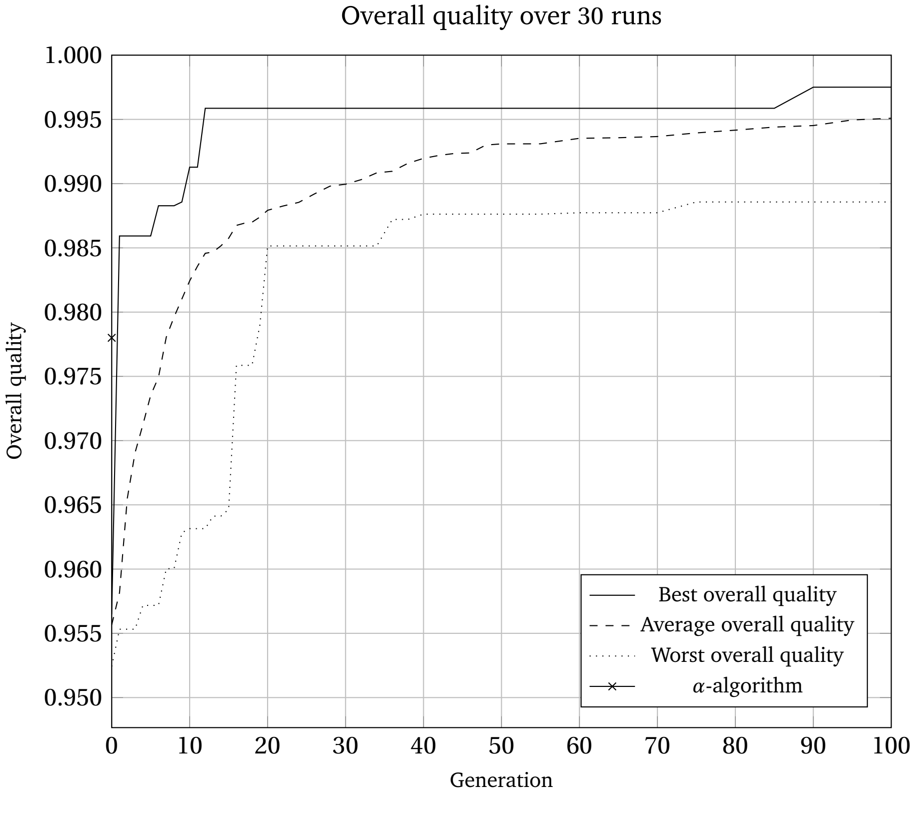
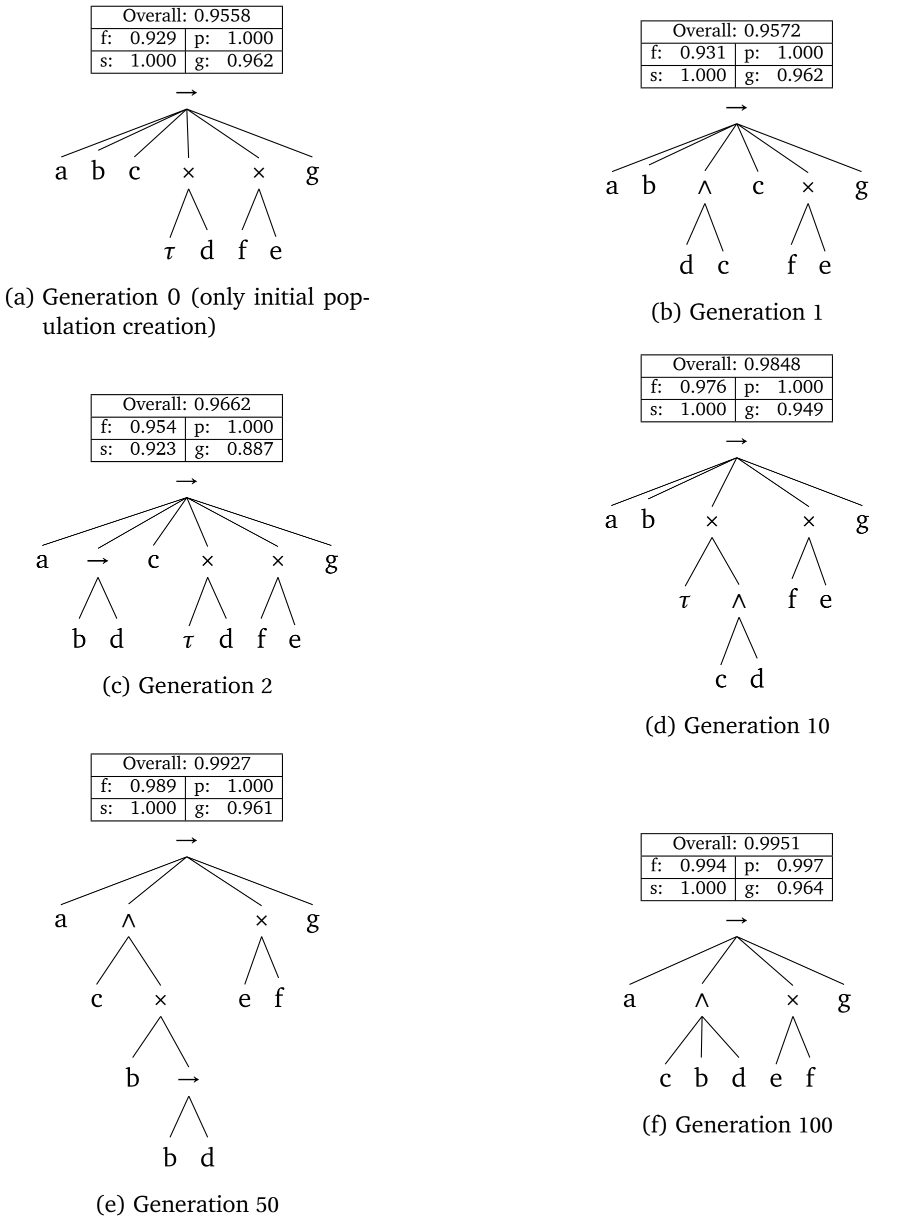


Figure 7.7: Minimum, average and maximum overall quality of the best candidates over 30 runs of ETM*d* on the running example with exceptional behavior, the first 100 generations. Behavior of the ETM*d* algorithm is more volatile in early generations.

already discovered than the best result of the other proces discovery algorithms (see Table 6.3).

The average process trees for a selection of generations are shown in Figure 7.8. The process tree in generation 0 is the same as for the running example without exceptional behavior. However, because of the exceptional behavior the replay fitness score is reduced to 0.929 (from 0.930). On the other hand, since there is more observed behavior, generalization increased from 0.880 to 0.962. Until generation 10 the discovered process trees are similar to the ones discovered for the running example without exceptional behavior. As of generation 50 it can be observed that the ETM*d* algorithm tries to incorporate the additional behavior, hence different process trees are discovered. The candidate of generation 50 for instance allows activity d to be skipped, resulting in high precision and replay fitness. The best process model in generation 100 however has a more generic process model, with higher replay fitness at the cost of precision. This process tree is the same as the final process model found for the running example without exceptional behavior, as can be seen in Figure 7.5a. However, for this event log, the process model can be further fine-tuned, increasing both replay fitness and precision, until the process model is found as is shown in Figure 7.5b.



(a) Generation 0 (only initial population creation)

(b) Generation 1

(c) Generation 2

(d) Generation 10

(e) Generation 50

(f) Generation 100

Figure 7.8: Process trees discovered by the ETM*d* algorithm on the running example with exceptional behavior in different generations. The process trees shown are closest to the average overall quality of all runs.

7.1.3 Pareto Front Evolution on Running Example with Exceptional Behavior

As discussed in Section 6.8.2, instead of weighting the different quality dimensions, the ETM*d* algorithm is also able to discover a Pareto front of candidates. We applied the ETM*d* algorithm, using the same settings as before, on the running example event log with exceptional behavior. This time, instead of considering a weighted overall quality, we construct a Pareto front of process trees.

Figure 7.9 shows the development of the size of the Pareto front for the 30 runs up to 10,000 generations. This graph shows that the number of candidates in the Pareto front very rapidly increases in early generations. In later generations the Pareto front size steadily keeps increasing to on average over 2,000 candidates after 10,000 generations. This indicates that the Pareto front becomes more detailed, i.e., more intermediate candidates are discovered. The graph also shows that, especially in early generations, the Pareto front size is sometimes reduced. This means that candidates are found that dominate candidates currently in the Pareto front. Although this results in a reduction of the size of the Pareto front, the quality of the discovered candidates is improved.

We selected the Pareto front with the size closest to the average size for further investigation. This Pareto front contains 1,996 candidates. Although the Pareto front contains a lot of candidates, only a few of these have enough quality in all quality dimensions to be considered. This is shown in Figure 7.10 which visualizes the distribution of the almost 2,000 process trees over the values for the different quality dimensions. The main observation is that although most candidates have a good replay fitness and precision score, the scores for generalization and simplicity are bad. Most of the candidates with low generalization or simplicity are 'bloated' process trees (see Section 4.3.3), i.e., process trees that contain a lot of operators that do not contribute to the perceived quality of the process tree.

Therefore, we post-process the Pareto front to construct a new Pareto front with the normalized versions (see Section 6.2) of the process trees from the original Pareto front. Figure 7.12 shows a process tree contained in the original Pareto front, and the normalized version that is contained in the post-processed Pareto front. Adding useless nodes, mainly operator nodes with only one child, improves precision, and the process tree is therefore contained in the Pareto front. However, the process tree itself is not simple nor generalizing. The resulting process tree after normalization, and re-evaluation of the four quality dimensions, has the same score for replay fitness. Precision is reduced slightly, but simplicity and generalization have improved greatly.

The post-processed Pareto front is built by first creating a new empty Pareto front. To this Pareto front we add the normalized and re-evaluated versions of all process trees of the original Pareto front. This ensures that all process trees in the new Pareto front do not contain useless nodes. Applying this process to the Pareto front originally containing 1,996 candidates results in a Pareto front containing 74 candidates. The histograms for this Pareto front are shown in Figure 7.11. The resulting Pareto front has better generalization scores, although

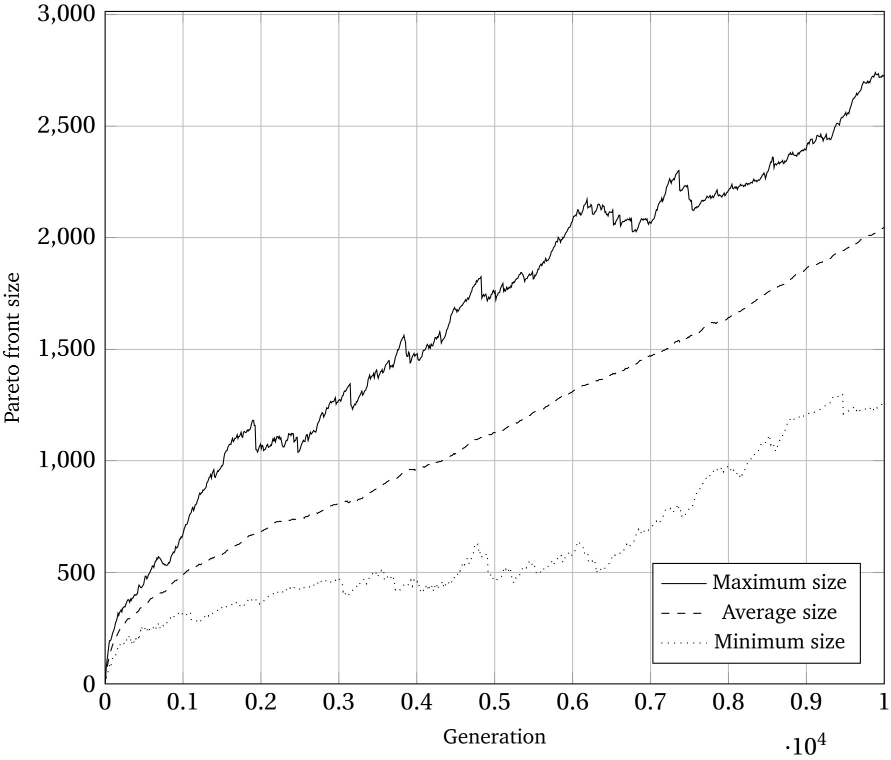


Figure 7.9: Minimum, average and maximum size of the Pareto front over 30 runs of the ETM*d* algorithm on the running example with exceptional behavior, all 10,000 generations. Overall the number of candidates in the Pareto front grows, although between generations the number of candidates might be reduced.

precision is somewhat reduced.

A visualization of this Pareto front on the quality dimensions of replay fitness, precision and generalization is shown in Figure 7.13. The goal is to discover a process tree that is positioned in the top-right corner with a bright yellow color, since this process tree would score perfectly (i.e., 1) on all three quality dimensions. Although this perfect candidate is not found, several candidates exist close to this point. A more detailed view of these candidates is shown in Figure 7.14 which focusses on the top-right grid cell of Figure 7.13. 35 candidates have a score for replay fitness and precision of at least 0.9. This shows that roughly half of the candidates are located in this section of the Pareto front.

Each candidate provides (slightly) different trade-offs between the three quality dimensions. Consider for instance the two pink candidates at the top, far right, which both have (close to) perfect precision. These candidates are visualized in Figure 7.15a and Figure 7.15d. This shows that, even though the difference in generalization is marginal, both process trees are contained in the

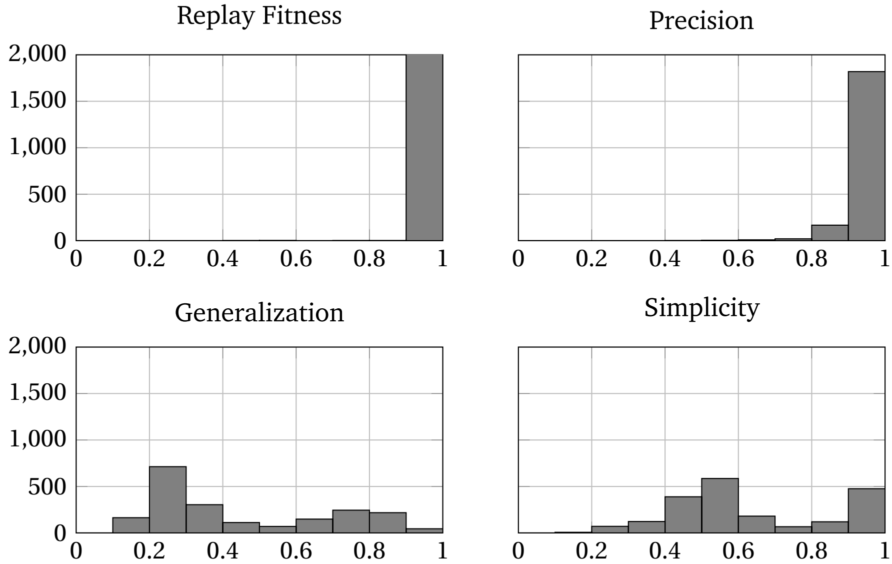


Figure 7.10: Distribution of the candidates over the quality dimensions in the Pareto front of 1,996 candidates. Most candidates have high replay fitness and precision scores but low scores for generalization and simplicity.

Pareto front. When we inspect the two models more closely we can see that the process tree shown in Figure 7.15a is the process tree that was discovered on the running example without exceptional behavior (see Figure 7.5a). However, this process tree does not describe all observed behavior, as is indicated by the replay fitness score of 0.994. The process with the best precision and perfect replay fitness is shown in Figure 7.15c. This process tree is however not very generalizing in the description of the behavior. A process tree with high replay fitness and precision, and better generalization, is shown in Figure 7.15d.

This experiment shows that constructing a Pareto front returns a (large) collection of process models to describe the observed behavior. However, not all candidates in the resulting Pareto front are useful descriptions of the observed behavior, for instance because many process trees contain unnecessary nodes.

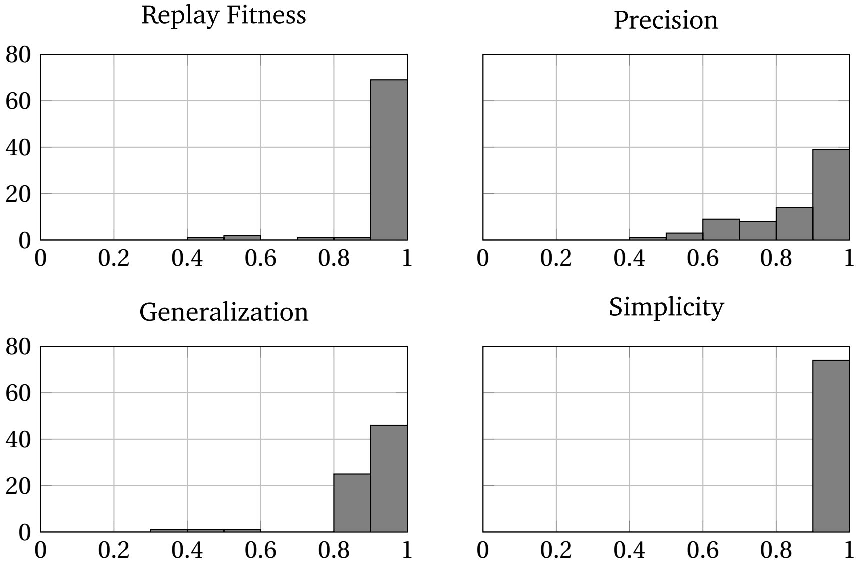
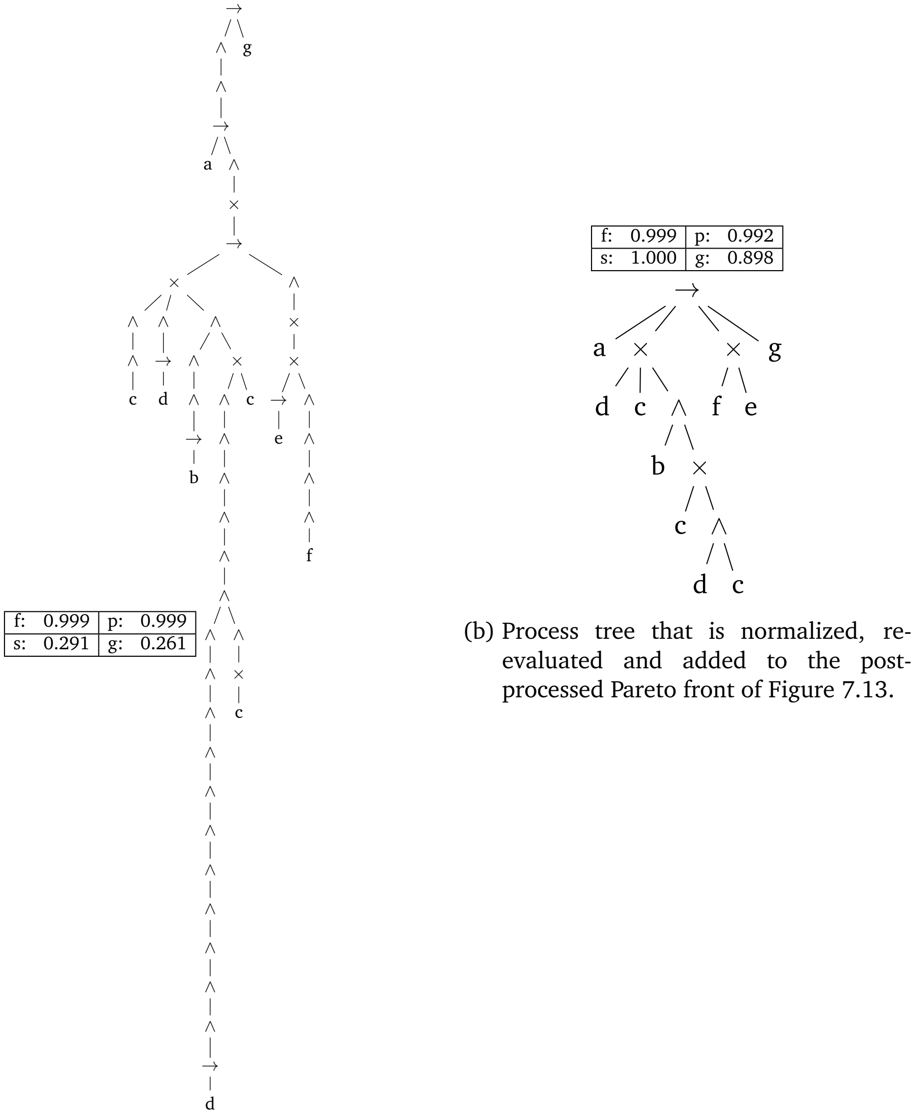


Figure 7.11: Distribution of the candidates over the quality dimensions in the normalized Pareto front containing 74 candidates. Simplicity and generalization improved while most candidates still have high replay fitness and moderate to high precision scores. Note that the y-scale changed between this figure and Figure 7.10, which also contained some candidates with medium precision scores.

Therefore the resulting Pareto front needs to be post-processed to remove these unnecessarily large process trees. This results in a smaller but more useful Pareto front of candidates. The candidates in the Pareto front can be further inspected and the different trade-offs in the quality dimensions can be evaluated by the end user.



(a) Process tree from the unprocessed Pareto front.

(b) Process tree that is normalized, re-evaluated and added to the post-processed Pareto front of Figure 7.13.

Figure 7.12: Example of process tree before (a) and after (b) post-processing which normalizes a process tree by removing useless nodes. Replay fitness remains the same but precision, generalization and simplicity change.

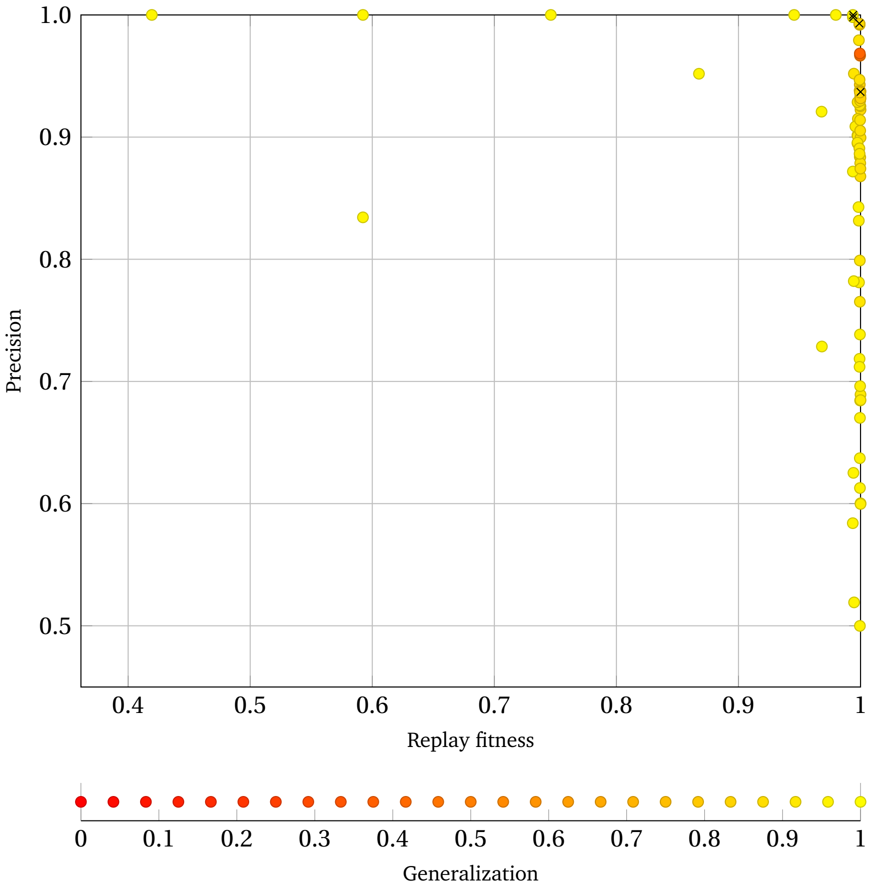


Figure 7.13: Visualization of the candidates in the normalized Pareto front. The Pareto front clearly shows the trade-off between replay fitness and precision. The ×-symbols indicate the four process trees shown in Figure 7.15.

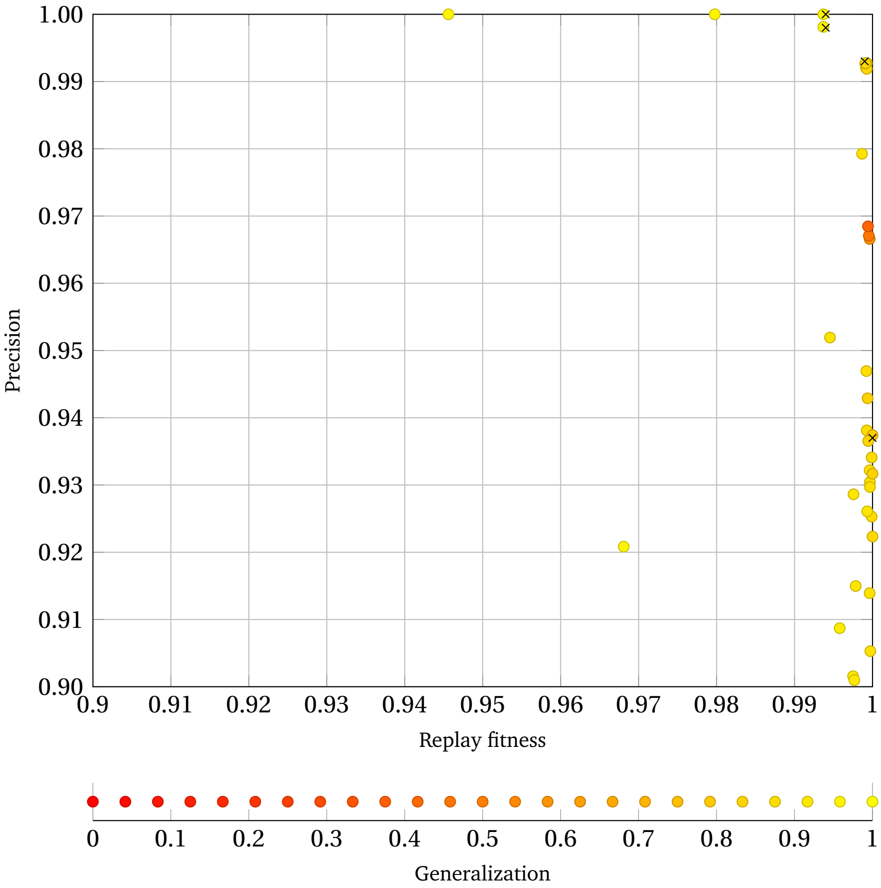
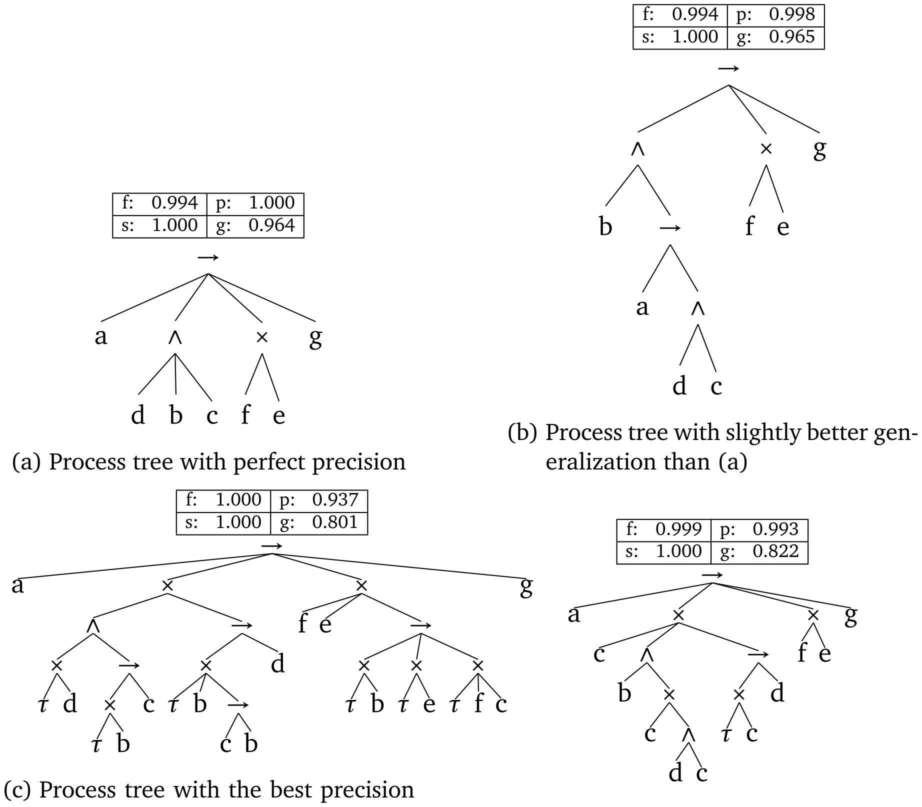


Figure 7.14: Visualization of the 35 candidates in the normalized Pareto front with both replay fitness and precision above 0.9. The fact that many models are still close together demonstrates that very small trade-offs are made. The ×-symbols indicate the four process trees shown in Figure 7.15.



(a) Process tree with perfect precision

(b) Process tree with slightly better generalization than (a)

(c) Process tree with the best precision for perfect replay fitness.

(d) Better generalization than (c)

Figure 7.15: Examples of process trees from the Pareto front of Figure 7.13 with slightly different trade-offs between the quality dimensions.

7.2 Random versus Guided Change

As discussed in Section 4.3 it is important to maintain enough variation in the population. Furthermore, the search needs to contain both explorative and exploitative aspects. Although guided operators contribute to the exploitative aspect of the search, they do not enable exploration of the search space. Therefore, they might not always be able to find the best possible candidate. This leads us to believe that random change operators are still required to introduce the diversity required to be able to discover good quality process trees. In this experiment we vary the ratio between the random and guided change operators applied. Figure 7.16 shows the average overall quality of the discovered candidates for different ratios of random change based on five runs per ratio.

From the graph it can clearly be observed that applying only random change operations or applying only guided change operations results in lower quality process trees being discovered. Applying only random change operators however shows gradual improvement over the generations. When only guided change operations are applied the growth stabilizes earlier, since guided change operators include little randomness. Not much (significant) difference can be observed between applying 25%, 50% or 75% random change operators. Although applying 75% random mutations seems to result in worse candidates in early generations, this effect is not significant.

The same experiment is run on the 'building permits - receipt phase' real life data set (which is investigated in more detail in Section 7.3). The resulting graph is shown in Figure 7.17. This graph shows that the overall quality of applying only random operators results in significantly worse process trees. The application of only guided change operators seems to result in good quality process trees, especially in earlier generations. However, it also quickly reaches its maximal overall quality, after which it does not show any further improvements. Applying 25%, 50% or 75% random change operators also shows a good performance and quickly results in good quality process trees. Moreover, these ratios also show gradual improvement of the overall quality in later generations.

Based on both these experiments we apply a random change operation ratio of 50% throughout this thesis. Especially for other applications of the ETM framework, where additional (and initially unknown) quality dimensions are considered, random change operations are required to construct high-quality process trees.

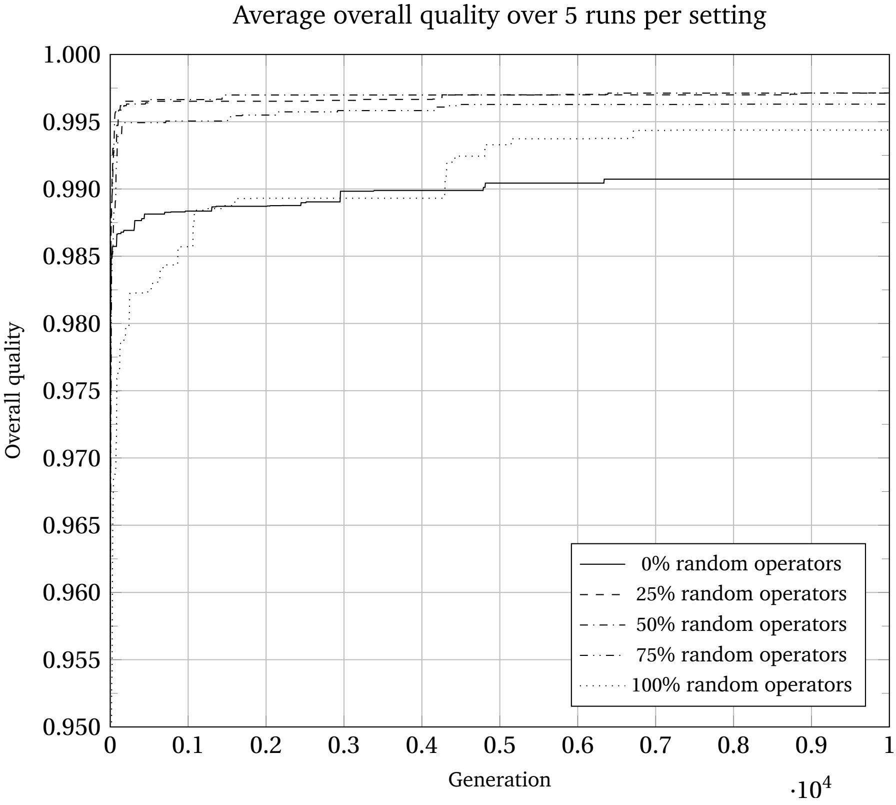


Figure 7.16: Comparison of different ratios of random mutation on the running example with exceptional behavior. For each ratio the average over five runs is taken. The best performance is achieved by a mixture of guided and random change behavior.

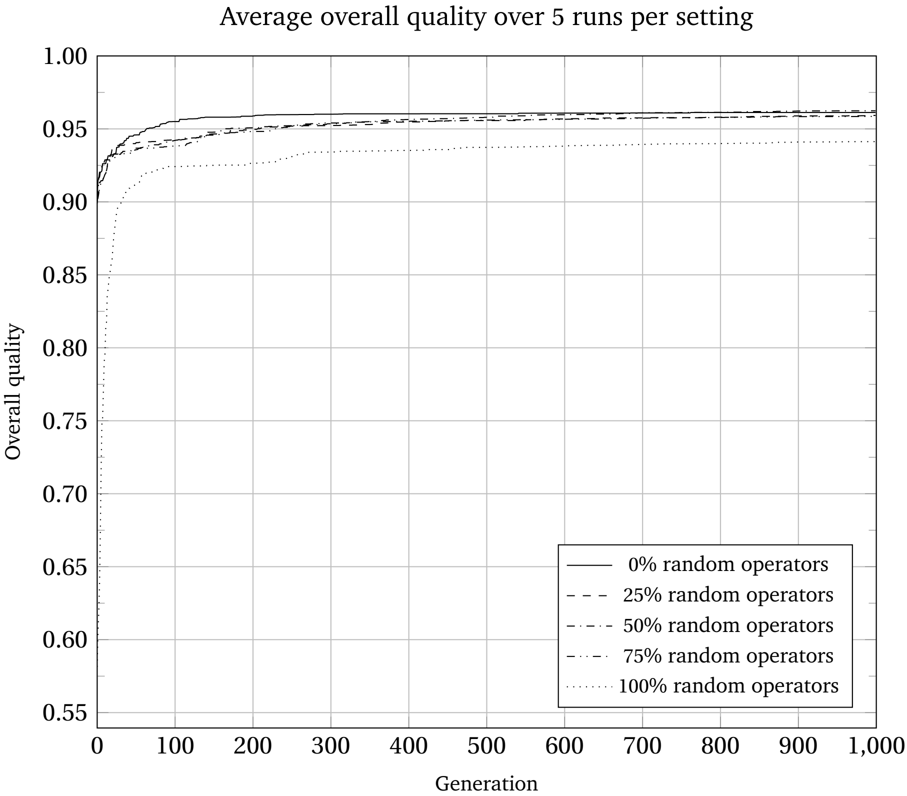


Figure 7.17: Comparison of different ratios of random change operators on the building permits event log over 1,000 generations. For each ratio the average over five runs is taken. The best performance in early generations is achieved by pure guided change, however a mixture of guided and random change does not perform significantly worse.

7.3 Building Permits Process - Receipt Phase

The 'building permits - receipt phase' event log describes the receipt phase of a building permits process. It contains 1,434 cases with in total 8,577 events and 27 different activities. Since the activity names are quite long, they are replaced by single letters (and 'aa'). A translation is shown in Table 7.2. Tasks have numbers, which indicate a pre-determined order and grouping of activities. Furthermore, activities a through d are all related to just handling the confirmation of receipt. It seems that this process is rather detailed with many checks and loop-backs to fix issues.

The ETM*d* algorithm was run for 1,000 generations in Pareto front mode, considering the four quality dimensions of replay fitness, precision, generalization and simplicity. The Pareto front was limited to only candidates with at least a score of 0.6 for replay fitness and precision, a score of at least 0.8 for simplicity, and to a maximum of 200 candidates. The resulting Pareto front was post-processed (as described in Section 7.1.3) to contain only process trees without useless nodes. The final Pareto front contains 104 candidates and is visualized in Figure 7.18.

The process tree with the best replay fitness is shown in Figure 7.19a. In order to obtain a high replay fitness, ↻- and ∨-operators are introduced. Therefore precision is low. The process model is not generalizing since many activities are present multiple times. An observation that can be made is that activity l (Confirmation of receipt) is always executed first. This is however followed by either no further action or by a very complicated proces.

The process tree with the highest score for generalization is shown in Figure 7.19b. Despite its high generalization, precision is higher than for the best replay fitness process tree. The process tree is also small and contains only six activities. However, replay fitness is low, since many frequent activities are not included. Again, activity l can be executed first, always followed by activity a (T02 Check confirmation of receipt). Furthermore, activities c (T04 Determine confirmation of receipt), d (T05 Print and send confirmation of receipt) and e (T06 Determine necessity of stop advice) are often executed together.

A process tree with a good trade-off between replay fitness and precision is shown in Figure 7.19c. Besides its high scores for replay fitness and precision, generalization is also high. However, only eight of the 27 activities are included. Given the high replay fitness score, this indicates that these eight activities occur very frequent, while the other activities occur less frequently. Again, activity l can be executed first, followed by activities a, c and d. There exists a loop of

Table 7.2: Activity codes and corresponding names for the building permits - receipt phase event log.

Event code	Event Name
a	T02 Check confirmation of receipt
b	T03 Adjust confirmation of receipt
c	T04 Determine confirmation of receipt
d	T05 Print and send confirmation of receipt
e	T06 Determine necessity of stop advice
f	T07-1 Draft intern advice aspect 1
g	T07-2 Draft intern advice aspect 2
h	T07-3 Draft intern advice hold for aspect 3
i	T07-4 Draft internal advice to hold for type 4
j	T07-5 Draft intern advice aspect 5
k	T08 Draft and send request for advice
l	Confirmation of receipt
aa	T09-1 Process or receive external advice from party 1
m	T09-2 Process or receive external advice from party 2
n	T09-3 Process or receive external advice from party 3
o	T09-4 Process or receive external advice from party 4
p	T10 Determine necessity to stop indication
q	T11 Create document X request unlicensed
r	T12 Check document X request unlicensed
s	T13 Adjust document X request unlicensed
t	T14 Determine document X request unlicensed
u	T15 Print document X request unlicensed
v	T16 Report reasons to hold request
w	T17 Check report Y to stop indication
x	T18 Adjust report Y to stop indication
y	T19 Determine report Y to stop indication
z	T20 Print report Y to stop indication

activity `e` (T06 Determine necessity of stop advice), which can be followed by the sequence of activities `f` (T07-1 Draft intern advice aspect 1) and `j` (T07-5 Draft intern advice aspect 5), which seem two related aspects. This is always followed by activity `p` (T10 Determine necessity to stop indication) which determines whether the process should be stopped.

We also applied other selected process discovery techniques to the event log. The results are shown in Figure 7.20. The Petri net discovered by the α-

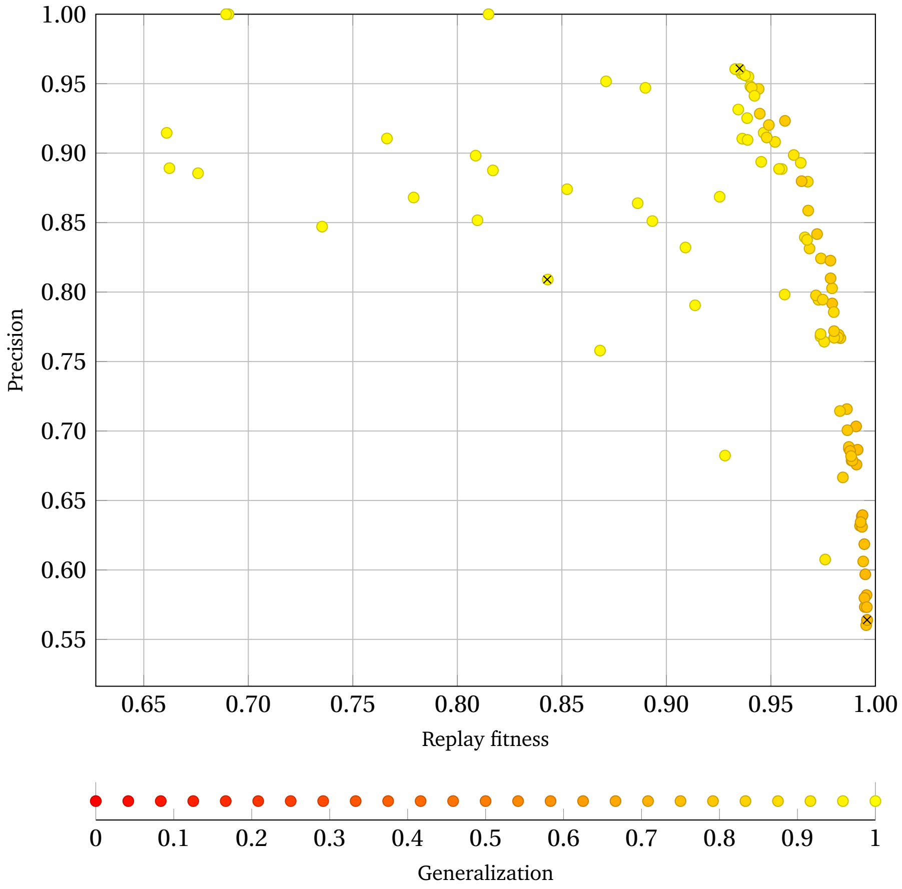


Figure 7.18: Visualization of the candidates in the Pareto front discovered for the building permits event log, where the color indicates the generalization value. The ×-symbols mark the three process trees that are visualized in Figure 7.19. Clearly trade-offs have to be made between replay fitness and precision. Especially high precision with reasonable replay fitness is hard to obtain.

algorithm (see Figure 7.20a) is large and complex. Moreover, the process model is unsound since the final marking cannot be reached. Therefore, replay fitness can only be calculated by special replay fitness metrics that can deal with not reaching a final marking [25]. The resulting replay fitness of 0.372 indicates that the Petri net discovered by the α-algorithm is not capturing the observed behavior at all. Moreover, precision is also low because there is a transition (in the top left) that is unbounded, i.e., this transition can always fire and produce two tokens each time.

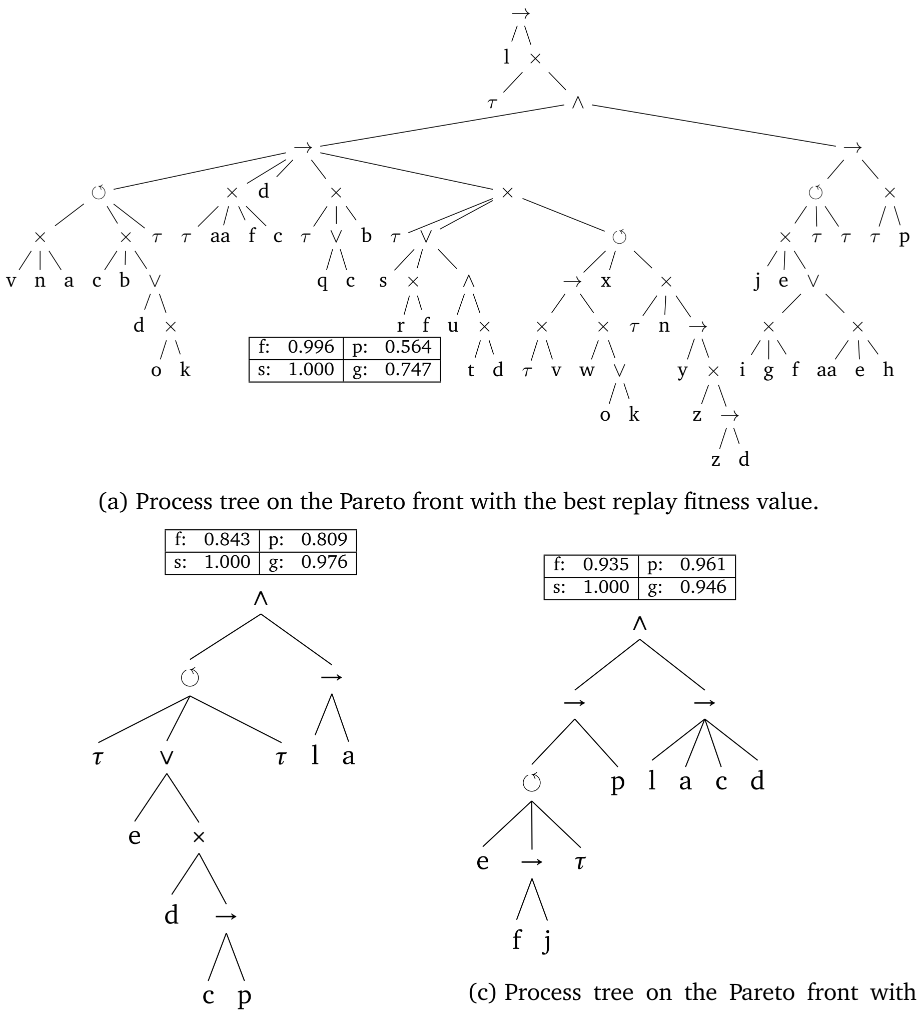
The Petri net translation of the causal net discovered by the Heuristics miner is shown in Figure 7.20b. This process model allows for a lot of behavior and has a replay fitness of 0.6626 (calculated on the Petri net using the work of [21,24]).

A process tree with perfect replay fitness is discovered by the Inductive Miner, as is shown in Figure 7.20c. This however comes at a severe cost of precision which is reduced to 0.278. The cause of this are the many $\circlearrowleft$-operators introduced to describe the different activities. It does however include every activity that occurs in the event log.

Note that the quality scores of all discovered process models shown in Figure 7.20 are outside the range of the Pareto front visualized in Figure 7.18. This indicates that the ETM*d* algorithm discovered superior process models.

Several observations can be made comparing the process models as discovered by the other process discovery algorithms with those in the Pareto front as constructed by the ETM*d* algorithm. First of all the ETM*d* algorithm is able to discover proces models with better quality, in all four dimensions. Moreover, the process models discovered by the ETM*d* algorithm are semantically correct, while this is not true for the α-algorithm and Heuristic miner algorithms. Finally, the translation of the discovered process tree to a BPMN process model results in a simple and structured process model.

This event log is used again in Chapter 8 where the ETM*r* algorithm is presented which is able to incorporate a normative process model.

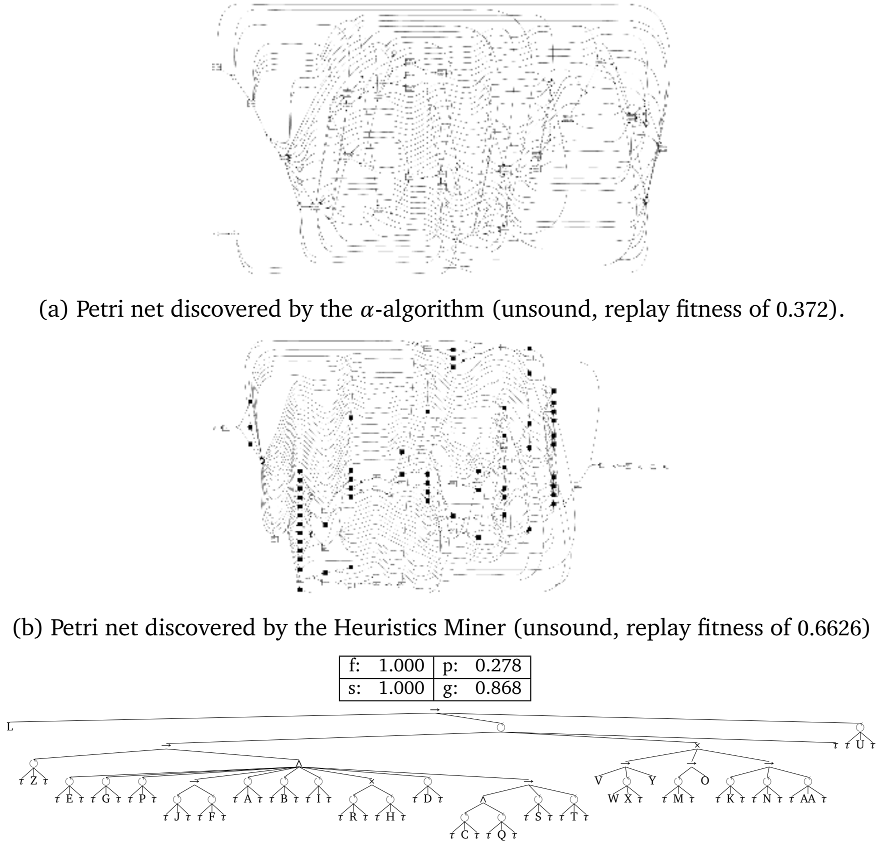


(a) Process tree on the Pareto front with the best replay fitness value.

(b) Process tree on the Pareto front with the best generalization value.

(c) Process tree on the Pareto front with a good trade-off between replay fitness and precision.

Figure 7.19: Process trees from the Pareto front for the building permits event log that provide different trade-offs between the four quality dimensions. High replay fitness is obtained by a big and imprecise process model, where smaller process models provide a more precise description with lower replay fitness.



(a) Petri net discovered by the α-algorithm (unsound, replay fitness of 0.372).

(b) Petri net discovered by the Heuristics Miner (unsound, replay fitness of 0.6626)

f:	1.000	p:	0.278
s:	1.000	g:	0.868

(c) Process tree discovered by the Inductive Miner

Figure 7.20: Results of the α-algorithm, Heuristics miner and Inductive Miner on the ‘building permits - receipt phase’ event log. The results of the α-algorithm and Heuristics miner have low replay fitness scores, while the result of the Inductive Miner has a very low precision.

7.4 Building Permits Process - Objections and Complaints

In this section we apply the ETM*d* algorithm on a collection of event logs obtained in the CoSeLoG project. The event logs contain activities related to handling objections and complaints regarding building permits from five municipalities. Basic characteristics of the five 'WABO' event logs are shown in Table 7.1 on page 164.

In order to save space and improve readability of the discovered process models, we use the codes used internally by the municipalities. A translation from these codes used in the process models to the names of the events is shown in Table 7.3. The codes in the 500 ranges are related to objections, in the 600 range to preliminary verdicts and in the 700 range to actual court decision. Note that a complaint or objection might be resolved during the process, so some cases never actually go to court.

The ETM*d* algorithm was run using exactly the same settings as before on each of the five event logs in isolation. The resulting Pareto fronts, after normalization of the candidates, are shown in Figure 7.21. The sizes of the discovered Pareto fronts vary between 95 candidates for event log WABO1_BB, to 116 candidates for event log WABO5_BB.

When we compare the Pareto fronts based on their visualizations, several observations can be made. The Pareto front for WABO4_BB for instance contains good quality process trees, as is indicated by the many dots close to the top-right corner. The Pareto fronts of WABO1_BB and WABO2_BB also are close to the top-right corner, while the Pareto fronts for WABO3_BB and WABO5_BB are further removed. The Pareto fronts for WABO1_BB and WABO3_BB contain more process trees with lower generalization than the other three Pareto fronts. This can be explained by the number of traces in the event logs, where fewer traces results in a lower generalization score.

From each Pareto front a process model is selected with a good balance of replay fitness and precision, and a relatively high generalization score. These selected models are shown in Figure 7.22. A first observation is that the process models all look significantly different. For event logs WABO1_BB, WABO2_BB and WABO3_BB activity 770 (Establish decision phase original decree) is added in parallel to the rest of the process. For event logs WABO4_BB and WABO5_-BB this activity is modeled in a more specific part of the process. Furthermore, activities 630 (Appeal set) and 730 (Contested disposal affected) are often executed closely together, although the activities seem to have little in common.

Table 7.3: Activity codes and corresponding names for the 'WABO' event logs.

Event code	Event Name
540	Objection to disposal submitted
550	Treat objection
550_1	Treat objection subcase
550_2	Treat objection subcase finished
560	Objection wrapped up
590	Received request for preliminary verdict
600	Treat preliminary verdict
610	Preliminary verdict wrapped up
630	Appeal set
640	Received request for preliminary verdict
670	Treat appeal
680	Appeal wrapped up
700	Higher objection started
730	Contested disposal affected
740	Verdict given by court
760	New decision or new evaluation
765	Phase start 2
766	New decision or new evaluation
770	Establish decision phase original decree
775	Decision phase definite
780_1	Create decree for the purpose of the disposal of the court
780_2	Connect disposal court
780_3	Register date of disposal of court
790	Establish decision phase of the verdict of court

Variations in the quality scores can also be observed, mainly in the scores for generalization. Although the process models appear to be very different, the processes should have similarities. Therefore, in Chapter 9 we present the ETMc algorithm to discover one process model to describe all five event logs.

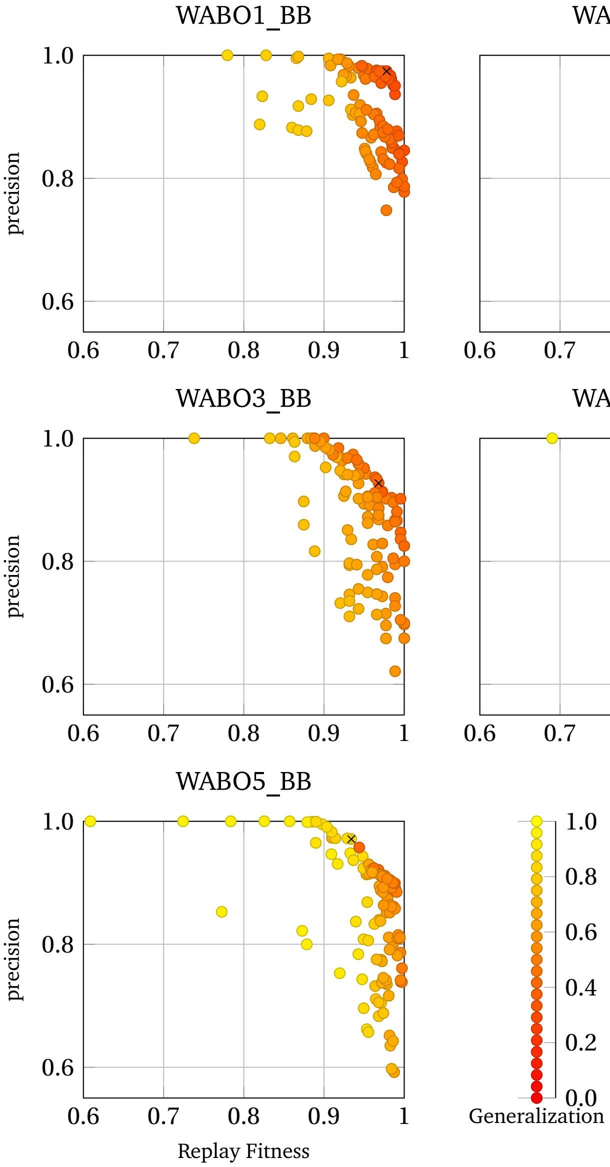
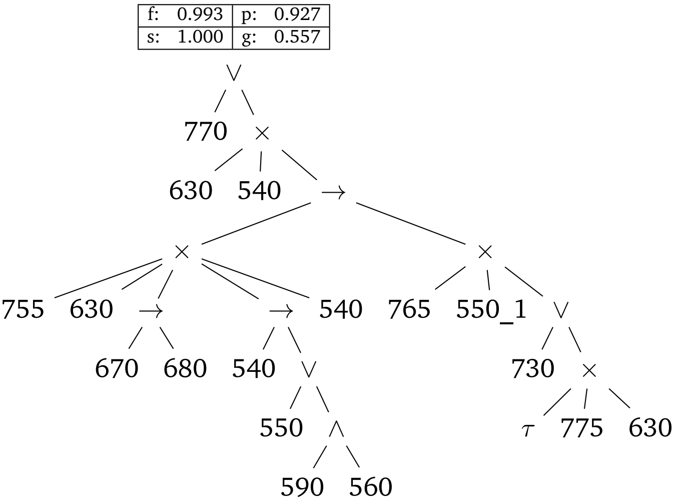
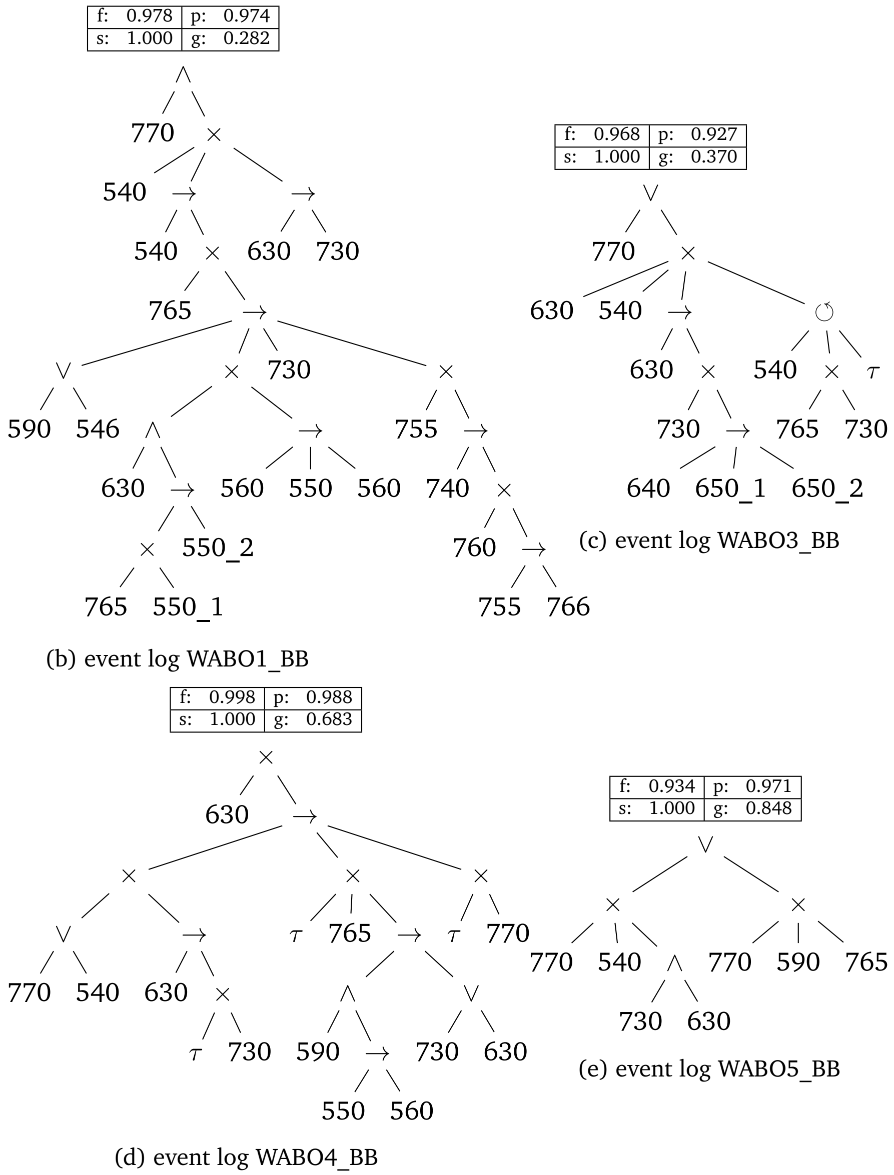


Figure 7.21: Visualization of the candidates in the Pareto front discovered for the WABO event logs, where the color indicates the generalization value. The ×-symbols mark the process trees that are visualized in Figure 7.22. Comparing the different Pareto fronts shows that trade-offs between replay fitness and precision have to made.



(a) event log WABO2_BB

Figure 7.22: Process trees for each of the event logs with a good balance of replay fitness and precision with high generalization (other event logs follow on next page).



(b) event log WABO1_BB

(c) event log WABO3_BB

(d) event log WABO4_BB

(e) event log WABO5_BB

Figure 7.22: (continued) Process trees for each of the event logs with a good balance between replay fitness and precision with high generalization (WABO2_BB on previous page).

7.5 Performance of the ETM*d* algorithm

The main advantage of evolutionary algorithms is the great flexibility offered to obtain high quality solutions under uncertain circumstances. However, this comes at the cost of performance, since evolutionary algorithms in general are slower than algorithms tailored towards specific problems [77, 90, 134]. Although evolutionary algorithms are applicable to a wide range of problems, problem-tailored methods are likely to perform better on specific types of problem.

As discussed in Section 4.4.1, the evaluation of candidates is usually the most time-consuming phase of an evolutionary algorithm. This is especially true for the ETM*d* and derived algorithms. It is estimated that 99% of all computation time is spent on calculating the replay fitness score, and in particular obtaining the optimal alignments between a process tree and the event log. Calculation of alignments is already heavily optimized in [21], and has been further improved by exploiting the properties of process trees. The fact that runs of the ETM*d* are relatively slow makes it more relevant to understand factors influencing performance. Therefore we discuss the time required for the ETM*d* algorithm to perform the experiments discussed in this chapter.

The experimental results presented in this chapter were obtained by running the ETM*d* algorithm on a collection of dedicated machines. Experiments were run on servers with 8 processor cores each (we did not use the hyperthreading virtual cores), each clocked at 2 Ghz. Each server contained 12 GB memory and ran Fedora 14 (64-bit). During the execution detailed statistics of the ETM*d* algorithm were logged in files. Besides statistics regarding the current population and Pareto front, performance statistics were also recorded.

Figure 7.23 shows the minimum, maximum and average time in seconds required per generation for the ETM*d* algorithm with weighted averages, as discussed in Section 7.1. Each generation took on average 0.2 seconds for the running example without exceptional behavior, and 0.4 seconds for the running example with exceptional behavior. Although execution times are relatively stable, some generations take longer. The graph of Figure 7.24 shows the time required to construct a Pareto front for both the running example with exceptional behavior, and the building permits process of Section 7.3. The time per generation steadily increases for the running example. This can be explained by the fact that the Pareto front was not limited in size or minimum quality of the process models. Especially big and imprecise process trees increase the time required to calculate the alignments of the process trees with the event log. The graph for the building permits process shows that limiting both the size

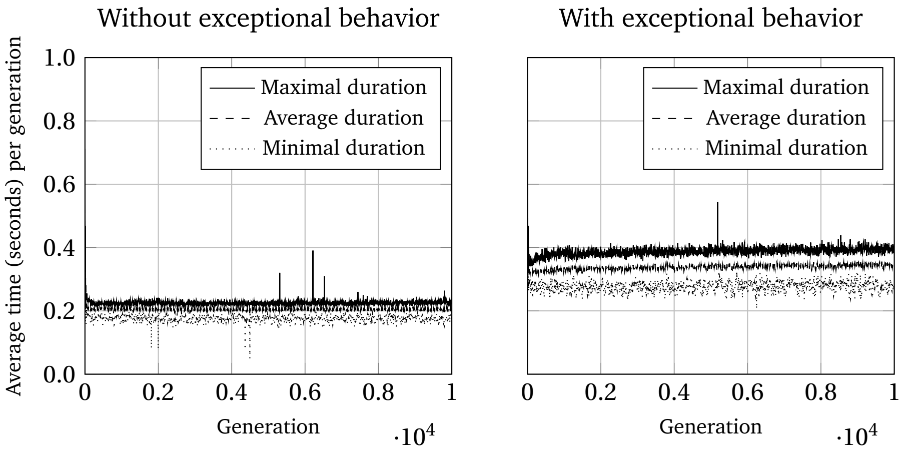


Figure 7.23: Performance statistics per generation for the running example. The duration per generation remains relatively constant.

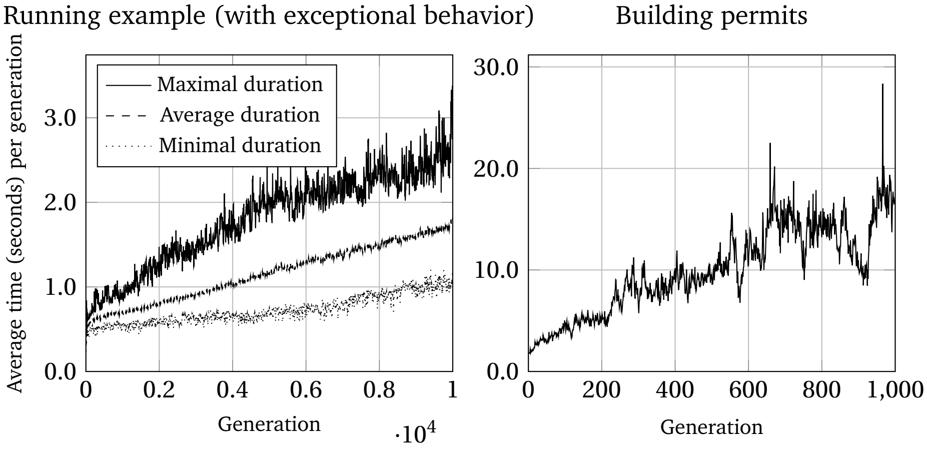


Figure 7.24: Performance statistics per generation while constructing a Pareto front. The time required per generation steadily increases.

and minimum quality of the process trees in the Pareto front does not stabilize the average time required for a generation. The average time per generation steadily increases. This real life event log also shows that the time required for a single generation can vary significantly between generations. While the average time per generation is 10 seconds, the maximum time observed is almost 28 seconds.

A performance evaluation of the ETM*d* algorithm while constructing a Pareto front for the five building permits event logs, as shown in Figure 7.25, shows that in general the performance is stable over time. The behavior as observed for the other building permits process (as shown in Figure 7.24) is only seen for the WABO 4 event log, which peaks to 10 seconds maximum per generation (not drawn in the chart), while the average is around 0.5 seconds. These graphs also show that the performance of the ETM*d* algorithm depends on many factors. The number of (unique) traces and the number of distinct activities have the largest influence on performance. This can be seen by comparing the performance of the largest event log, WABO 5, to the other WABO event logs. The performance of the ETM*d* algorithm on WABO 5 is more irregular than on the other WABO event logs. The characteristics of this event log are similar to the WABO 2 event log regarding size and number of events. Therefore, other characteristics, such as the underlying process of the event log, have influence on the performance, where more structured processes result in simpler process models.

As mentioned before, the performance of the ETM*d* algorithm mainly depends on the calculation of the alignments. Alignment calculation has already been heavily optimized and discussed extensively by Adriansyah [21]. Since alignment calculation is not the topic of this thesis, we do not address this any further. Therefore, we evaluate the performance of the ETM*d* algorithm, and derived algorithms, mainly on the number of generations required.

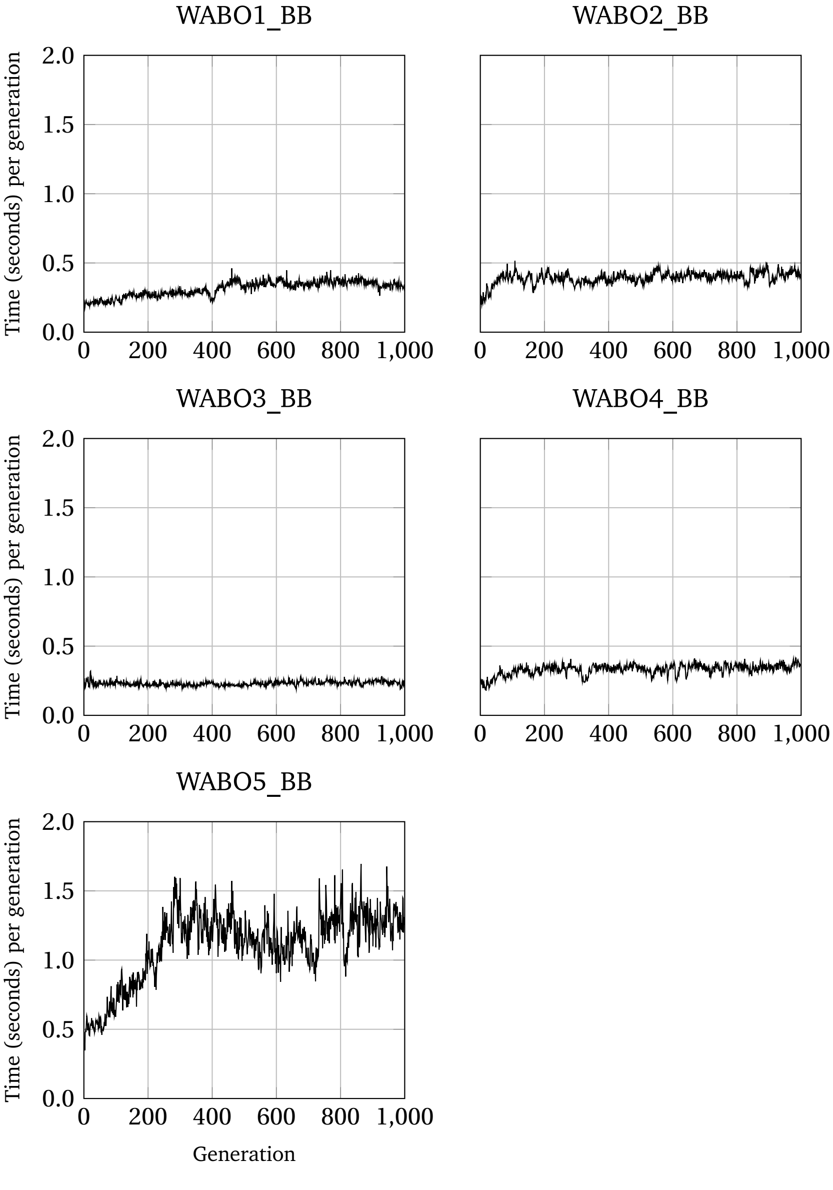


Figure 7.25: Performance statistics per generation for the WABO event logs. The time per generation is stable for event logs WABO1_BB through WABO4_BB. For WABO5_BB the time per generation is higher and less steady.

7.6 Conclusion

Applying the ETM*d* algorithm on both artificial and real life event logs demonstrated the importance of considering all four quality dimensions, and communicating these scores to the user.

In this chapter we applied the ETM*d* algorithm on both artificial and real life event logs. We have demonstrated that the ETM*d* algorithm is able to quickly discover process models of superior quality in comparison to other process discovery algorithms. This is especially true for real life event logs, as demonstrated by the results of current process discovery algorithms in Section 7.3. The analysis also showed that although quality increases quickly in early generations, improving the discovered process models becomes harder in later generations.

This chapter also demonstrated the actual process trees contained in a Pareto front. After post-processing, process trees of high quality remain in the Pareto front. This also demonstrates that, even for simple artificial event logs, different process models can describe the observed behavior. We have also shown that the Pareto front provides additional insights into real life event logs.

The experiments with applying different ratios of random change operators demonstrated that neither only random, nor only guided, change is a good idea. Although guided change operators use existing knowledge to improve the process model, randomness is still required to explore different parts of the state space.

We also analyzed the performance of the ETM*d* algorithm in this chapter. We showed that performance of the ETM*d* is stable during the run of the algorithm. An exception is when the ETM*d* algorithm is constructing a Pareto front without lower boundaries for the quality of the process models included. In this case the time per generation steadily increases because the Pareto front contains more and more low-quality process models which require long computation times. Therefore limiting the candidates allowed in the Pareto front increases the performance of the ETM*d* algorithm.

Chapter 8

Balancing Observed and Modeled Behavior

In this chapter we address Challenge 5, “Use Existing Knowledge in Process Discovery”, and present the ETM*r* algorithm. The system that supports a business process is usually configured using a (paper) process model. This means that for most event logs, a process model exists that also describes the observed behavior. However, differences between the modeled and observed behavior might exist. The description might not be accurate since the system might allow for deviations, or the implementation deviated from the documented process model. This given process model is currently mainly used for analysis purposes, e.g. comparing modeled behavior with observed behavior. Current analysis techniques can compare the given process model to observed behavior by evaluating the four quality dimensions of replay fitness, precision, generalization and simplicity. Furthermore, deviations of the observed behavior from the process model can be visualized on both the process model and the observed traces. However, given a process model and observed behavior, only few techniques exist that are able to repair this process model using the observed behavior. Much more insight into the observed deviations of the process model can be provided

Parts of this chapter are based on the work presented in [53]:
J.C.A.M. Buijs, M. La Rosa, H.A. Reijers, B.F. van Dongen, and W.M.P. van der Aalst. Improving business process models using observed behavior. In P. Cudre-Mauroux, P. Ceravolo, and D. Gašević, editors, *SIMPDA*, volume 162 of *Lecture Notes in Business Information Processing*, pages 44–59. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-40918-9.

by, instead of showing where deviations occur, showing how the process model can be changed to support these deviations. In this chapter we explicitly use the given process model during process discovery. This idea is also presented as one of the possible applications of the ETM framework in Section 4.2.2.

In this chapter we present the ETM*r* algorithm which, given an event log and one or more process models, is able to repair the given process model(s) while maintaining a certain similarity to the original process model. Section 8.1 discusses several application scenarios of the ETM*r* algorithm. This is followed by a discussion on measuring similarity in Section 8.2. We then apply the ETM*r* algorithm on the running example of Section 1.2.1 in Section 8.3. This is followed by an application on a case study data set from the CoSeLoG project in Section 8.4. In Section 8.5 related work in the area of process model repair is discussed. Section 8.6 concludes this chapter.

8.1 Application Scenarios

Given a process model and an event log, deviations can be visualized on both the process model and the traces in the event log. However, the process model can also be updated using the observed behavior, in essence performing *process model repair* using observed behavior. By comparing the modeled and observed behavior, changes can be made to the process model to better describe the observed behavior. Activities can be added or removed, and relationships between activities can be changed. By allowing less similarity to the input process model(s) a more rigorous repair can be performed, thus increasing the scores on the four quality dimensions.

When the similarity to the input process model(s) is relaxed, more freedom is allowed when repairing the process model. However, the resulting process model is likely to still have a similar overall structure to the original process model. Moreover, since behavior can be modeled and discovered using many different process modeling constructs, by considering the input process model the discovered process model is more similar to the process model as it is known within the organization. This can also *help in adopting and understanding the discovered process model*, since it is similar to the process model known within the organization.

An additional scenario could be the comparison of a previously discovered process model with a more current event log. By using the previously discovered process model as input, this model is repaired using a more recent event log. This indicates what has changed in the behavior between the two event logs.

This can be concept drift, change in laws or customer behavior, etc. Although this approach does not explain why something changed, or exactly when, it does show what has changed which aids further investigation.

8.2 Similarity as the 5th Quality Dimension

In order to extend the ETM*d* algorithm to support process model improvement we only need to add a metric that measures the similarity of the candidate process model to the reference process model. Similarity of business process models is an active area of research [18, 63, 68, 104, 115, 118, 124–126, 187]. We distinguish two types of similarity: *behavioral similarity* and *structural similarity*. Approaches focusing on behavioral similarity, e.g. [18, 63, 68, 115, 187], encode the behaviors described in the two process models to compare using different relations. Examples are causal footprints [68], transition adjacency relations [187], and behavioral profiles [115]. By comparing two process models using such relations, it is possible to quantify behavioral similarity in different ways.

Approaches focusing on structural similarity only consider the graph structure of models and abstract from the actual behavior. Heuristic approaches like [124–126] reduce the process model to an order matrix and calculate the difference from that representation. Most approaches [63, 104, 118] provide a similarity metric based on the minimum number of *edit operations* required to transform one model into another model. An edit is an atomic operation that inserts or removes either a node or an arc.

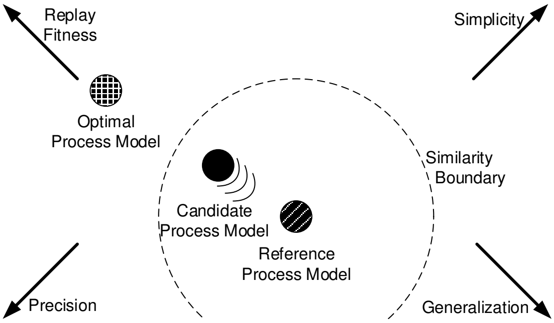


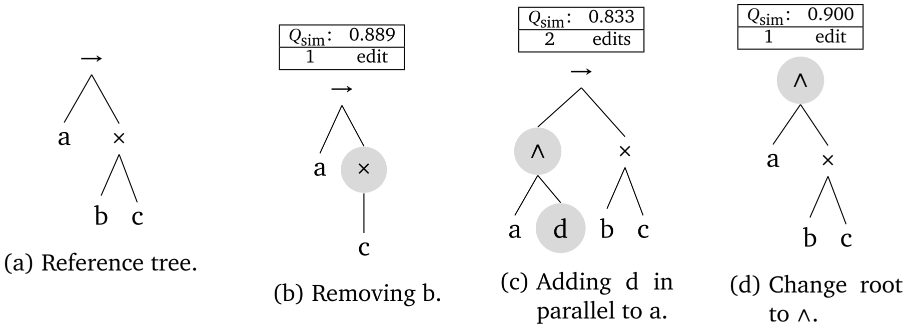
Figure 8.1: Similarity as the 5th quality dimension, influencing the other 4 quality dimensions.

Both behavioral and structural similarity approaches first require a suitable *mapping* of nodes between the two models. This mapping can be achieved by combining techniques for syntactic similarity (e.g. using string-edit distance) with techniques for linguistic similarity (e.g. using synonyms) [63].

*The ETM*r *algorithm however only needs to consider the structural similarity between candidate process trees and the reference process model.* The event log already captures the behavior that the process model should describe, and the goal is to improve the process model to better describe the observed behavior. Recall that the behavior of the reference model with respect to the logs is already measured by means of three of the four process discovery quality dimensions (replay fitness, precision and generalization). Hence, we only use structural similarity to quantify the fifth dimension.

Since we use process trees as our internal representation, similarity between two process trees can be expressed by the tree edit distance for ordered trees [147]. The tree edit distance indicates the minimum number of simple edit operations (add, remove and change) that need to be made to nodes in one tree in order to obtain the other tree. For process trees, each edit adds, removes or changes one node in the tree. Adding a node can for instance mean to add a node as a child of an existing node, or in between two existing nodes (in between a parent and its child). Removing a single node from the tree counts as one edit, hence removing a subtree counts as removing all individual nodes in that subtree. Furthermore, the type of a node can be changed, which also counts as one edit.

In the case of a weighted overall quality, the other four quality metrics are normalized to values between 0 and 1. Therefore we need to do the same for



(a) Reference tree.

(b) Removing b.

(c) Adding d in parallel to a.

(d) Change root to ∧.

Figure 8.2: Examples of possible edits on a tree (a) and respective similarities.

the edit distance. Thus, the similarity, using the edit distance Q_{sim}, is calculated as follows:

$$Q_{\text{sim}} = 1 - \frac{\#\text{edits}}{\#\text{nodes in reference model} + \#\text{nodes in candidate}} \tag{8.1}$$

Hence, a similarity score of 1.000 means that the process model is the same as the reference model. In case the similarity score is 0.000, this means that there was no overlap whatsoever between the two process trees. Please note that this is an extreme case, and in general change operations prevent this score (since one change is equivalent to adding and removing one node). In case the ETM*r* algorithm constructs a Pareto front, the number of edits is used instead of the normalized edit distance.

Figure 8.2 shows examples for each of the three edit operations. The reference tree is shown in Figure 8.2a. Figure 8.2b shows the result of deleting activity b from the tree. This leaves the ×-operator with only one child. The removal of activity b from the tree results in an edit distance of 1, and hence the resulting similarity is $1 - \frac{1}{5+4} = 0.889$.

The process tree shown in Figure 8.2c has activity d added in parallel to activity a. This results in two edits since a new $\wedge$-operator node needs to be added, including a leaf for activity d. The similarity of 0.833 is lower than when part of the tree is deleted, since two nodes have been added. Finally, changing a node as shown in Figure 8.2d, where the root →-operator is changed into an $\wedge$-operator, only requires 1 edit operation and in this case results in a similarity of 0.900.

We use the Robust Tree Edit Distance (RTED) algorithm [147] to calculate the edit distance between two ordered trees. The RTED approach first computes the optimal strategy to use for calculating the edit distance. It then calculates the edit distance using that strategy. Since the overhead of determining the optimal strategy is minimal, this ensures the best performance and memory consumption, especially for larger trees. However, it is important to realize that our approach is not limited to the RTED algorithm, any similarity notion can be used. The only requirement is that this new metric used to express the similarity notion adheres to the metric requirements as discussed in Section 4.3.4.

8.3 Application to Running Example

Throughout this section we re-use the running example of Section 1.2.1 to explain our approach. We assume that next to the event log, shown in Table 8.1, there is also a process model known within the organization, as is shown in Figure 8.3. The organization wants to know how this proces model should be changed to better reflect the recorded behavior.

The process model of Figure 8.3 describes the following process flow. When a potential customer fills in a form and submits the request from the website, the process starts by executing activity a which notifies the customer of the receipt of the request. Next, according to the process model, there are two ways to proceed. The first option is to start with checking the credit (activity b) followed by calculating the capacity (activity c), checking the system (activity d) and rejecting the application by executing activity f. The other option is to start with calculating the capacity (activity c) after which another choice is possible. If the credit is checked (activity b) then finally the application is rejected (activity f). Another option is the only one resulting in executing e, concerned with accepting the application. Here activity d follows activity c, after which activity b is executed, and finally activity e follows. In all three cases the

Table 8.1: The event log

Trace	#
a b c d f g	38
a b d c f g	26
a b d c e g	12
a b c f g	8
a b c d e g	6
a d c b f g	4
a c d b f g	2
a c b e g	1
a d b c f g	1
a d b c e g	1
a c b f g	1

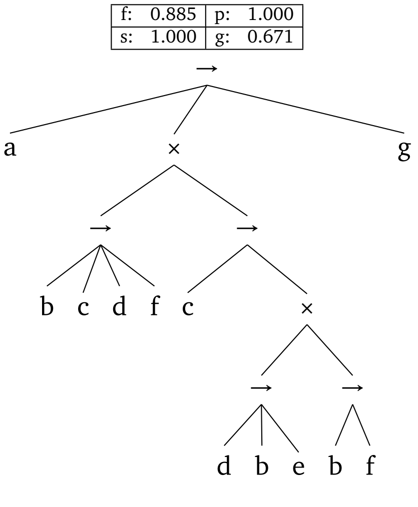


Figure 8.3: Process tree as known within the company.

process ends with activity g, which notifies the customer of the decision made.

However, the observed behavior, as is recorded in the event log shown in Table 8.1, deviates from this process model. The event log contains 11 different traces whereas the original process model only allows for 3 traces. The modeled and observed behavior thus differ significantly. To demonstrate the effects of considering the similarity between process trees, we run the ETM*r* algorithm on the example data shown in Table 8.1 and Figure 8.3. We construct a Pareto front of process models where the number of edits applied on the reference process model is used as the fifth quality dimension.

The ETM*r* algorithm constructed a Pareto front containing 573 candidates in 10,000 generations. Figure 8.4 shows the distribution of the process trees over the quality dimensions. The maximum number of edits applied to one of the candidates is 21. In general between 3 and 14 edits are applied. A two-dimensional view on the Pareto front is shown in Figure 8.5 where the trade-off between replay fitness and precision is shown. The colors indicate the number of edits applied. This demonstrates that most candidates have very high scores for precision and replay fitness. Views filtering different ranges of edits applied are shown in Figure 8.6. This demonstrates that applying only one edit already can improve the process tree significantly, considering that the normative process tree has a perfect precision and a score of 0.885 for replay fitness. Allowing more edits increases the quality of the process tree with regard to the original four quality dimensions. As can also be observed in Figure 8.4, most process trees have between 3 and 14 edits applied.

Figure 8.7 shows a selection of three process models from the Pareto front. The process tree shown in Figure 8.7a shows the process tree with the best replay fitness score, while maintaining perfect precision. This process tree can be obtained by applying 11 edits to the normative process tree. By applying 11 different edits to the normative process tree the model shown in Figure 8.7b can be obtained with perfect replay fitness, while still scoring high on precision. The process tree with the most edits is shown in Figure 8.7c. This process tree does not score best in any of the individual quality dimensions but provides a good trade-off between replay fitness and precision.

A side-effect of the required similarity is that the process tree of Figure 8.7c contains many useless nodes, which cannot be removed without increasing the edit distance. However, not all useless nodes in this process tree are present in the original process model. By adding useless nodes precision is improved since the fraction of escaping edges is reduced. Normalizing the process tree is not possible since removing certain useless nodes reduces the similarity with the reference model. Therefore we use the non-normalized Pareto front in this chapter.

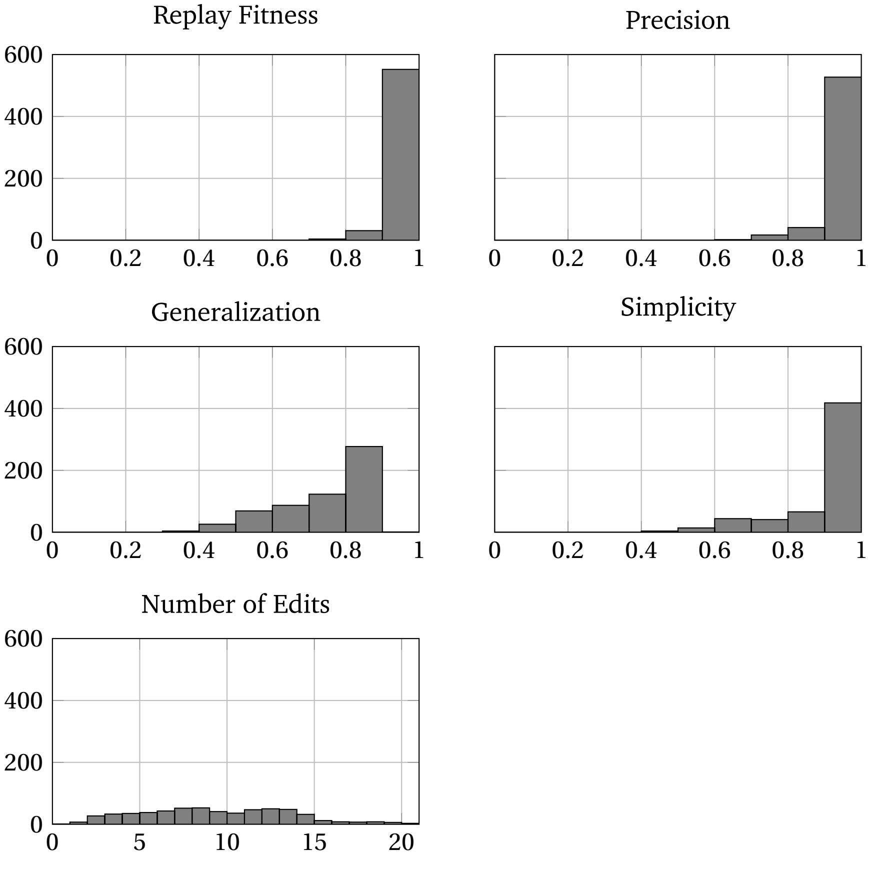


Figure 8.4: Distribution of the candidates over the quality dimensions in the Pareto front of 573 candidates for the running example.

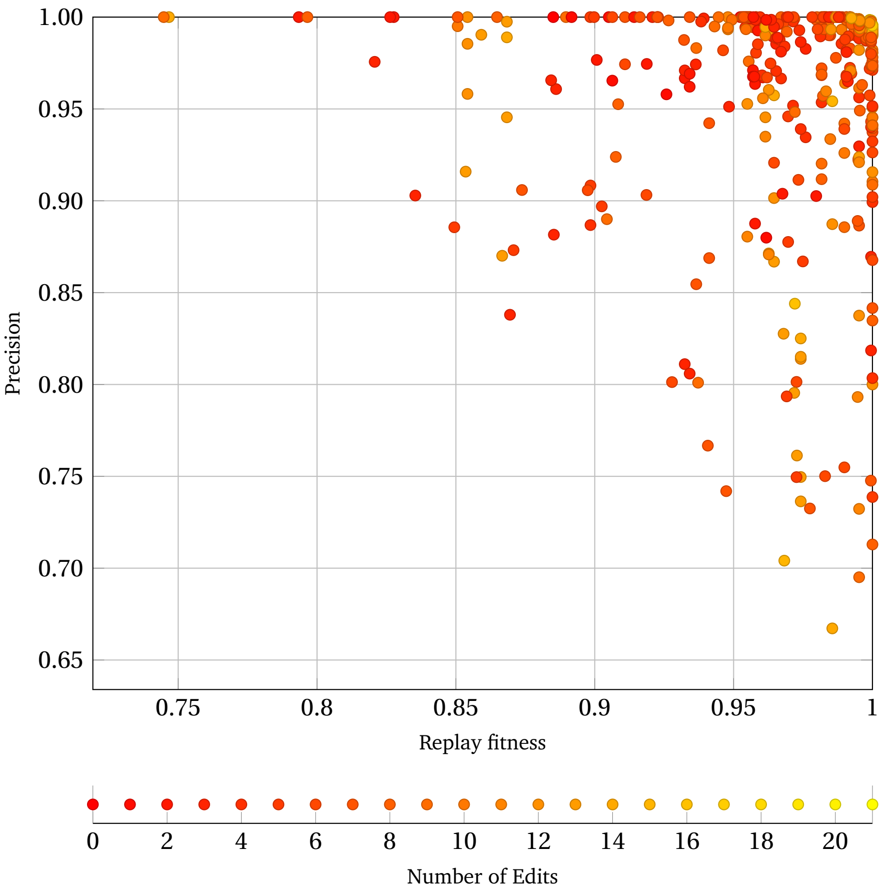


Figure 8.5: Visualization of the Pareto front of process trees discovered by the ETM*r* algorithm on the running example event log.

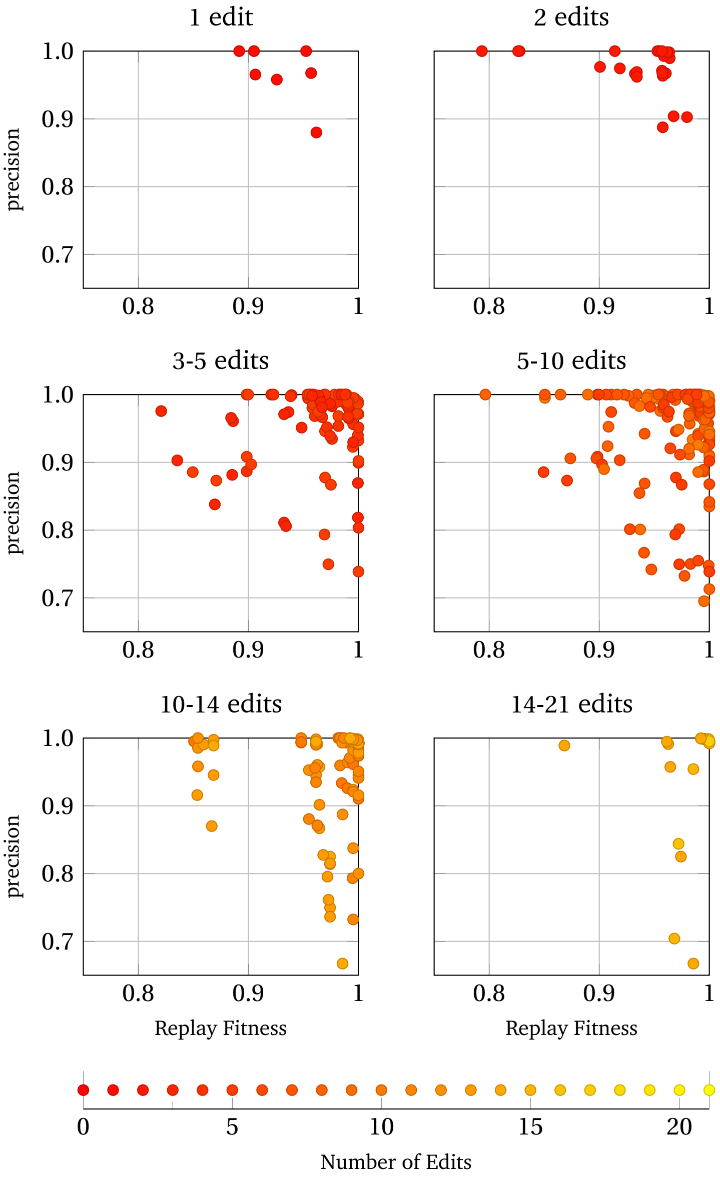
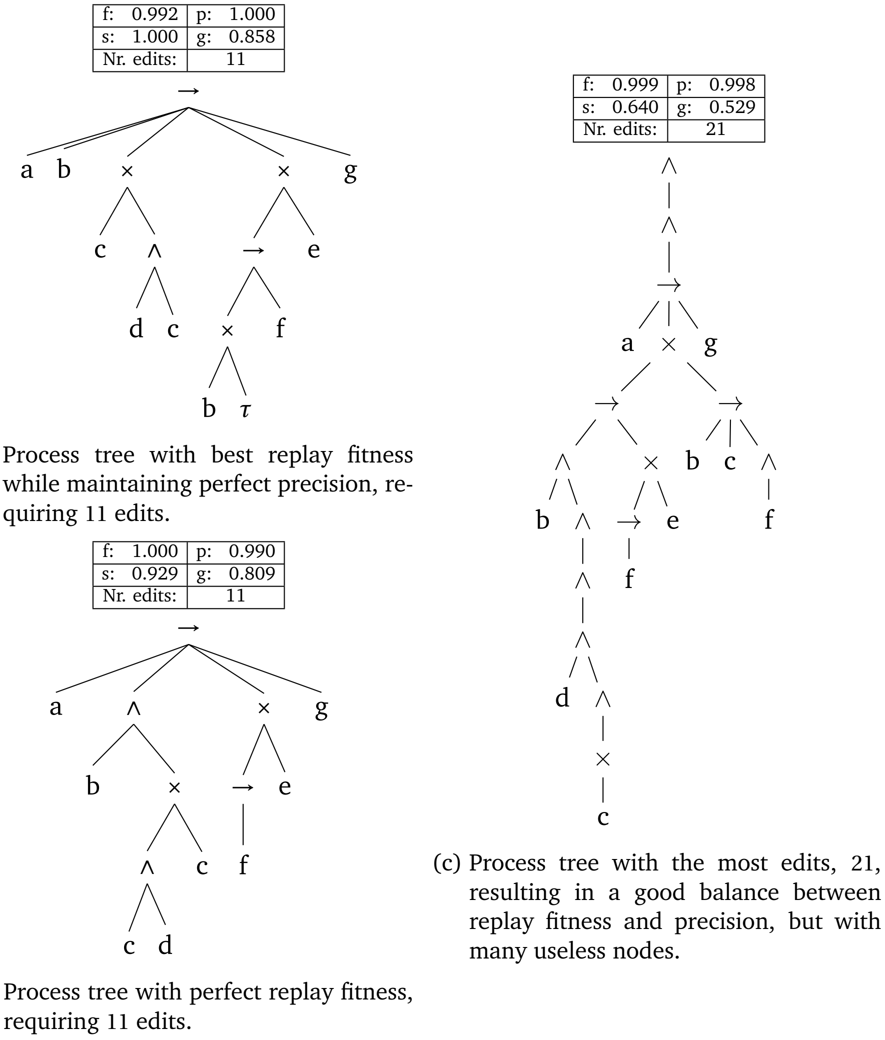


Figure 8.6: Visualizations of the Pareto front discovered by the ETM*r* algorithm on the running example, filtered by the number of edits allowed.



(a) Process tree with best replay fitness while maintaining perfect precision, requiring 11 edits.

(b) Process tree with perfect replay fitness, requiring 11 edits.

(c) Process tree with the most edits, 21, resulting in a good balance between replay fitness and precision, but with many useless nodes.

Figure 8.7: Process trees discovered while maintaining similarity with the normative model for the running example.

8.4 Case Study

One of the municipalities participating in the CoSeLoG project (see Section 1.4) implemented the process for receiving permits using a business process management (BPM) system. The BPM system was configured using a process model. The municipality is interested in comparing the described behavior with the observed behavior. The process tree representation of the process model used to configure the BPM system is shown in Figure 8.8. Activities have been relabeled to letters for readability, a translation is provided in Table 7.2 on page 187. The process model foresees that activity l (Confirmation of receipt) is executed first, followed by a complicated process, which includes loops, handling different aspects. The normative process model allows for improvements in both the replay fitness and precision quality dimensions.

The obtained event log contains 1,434 cases and 8,577 events representing 27 activities. The event log describes the process for the receipt phase of building

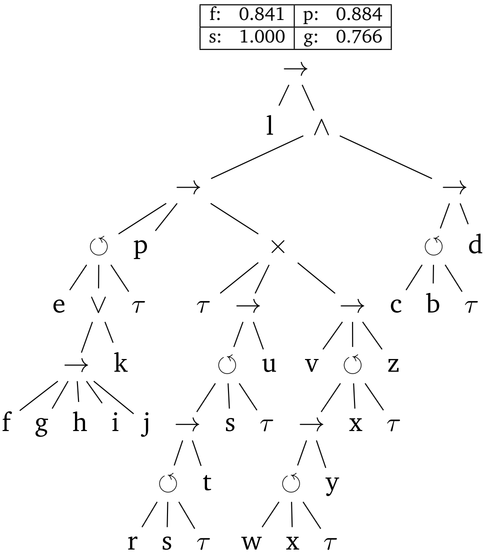


Figure 8.8: Process tree used as the normative model for the case study application of the ETM*r* algorithm. The meaning of the letters is defined in Table 7.2 on page 187.

permits. It consists of activities for notifying the applicant of the receipt of their request, but also activities for creating, testing and finalizing documents. Due to space restrictions, the activities are relabeled to single letters (and 'aa'). A more detailed discussion of the process, and an application of the ETM*d* algorithm on the same event log, can be found in Section 7.3.

The ETM*r* algorithm is run for 10,000 generations to construct a Pareto front that was limited to 200 process trees and scores of at least 0.6 for replay fitness,

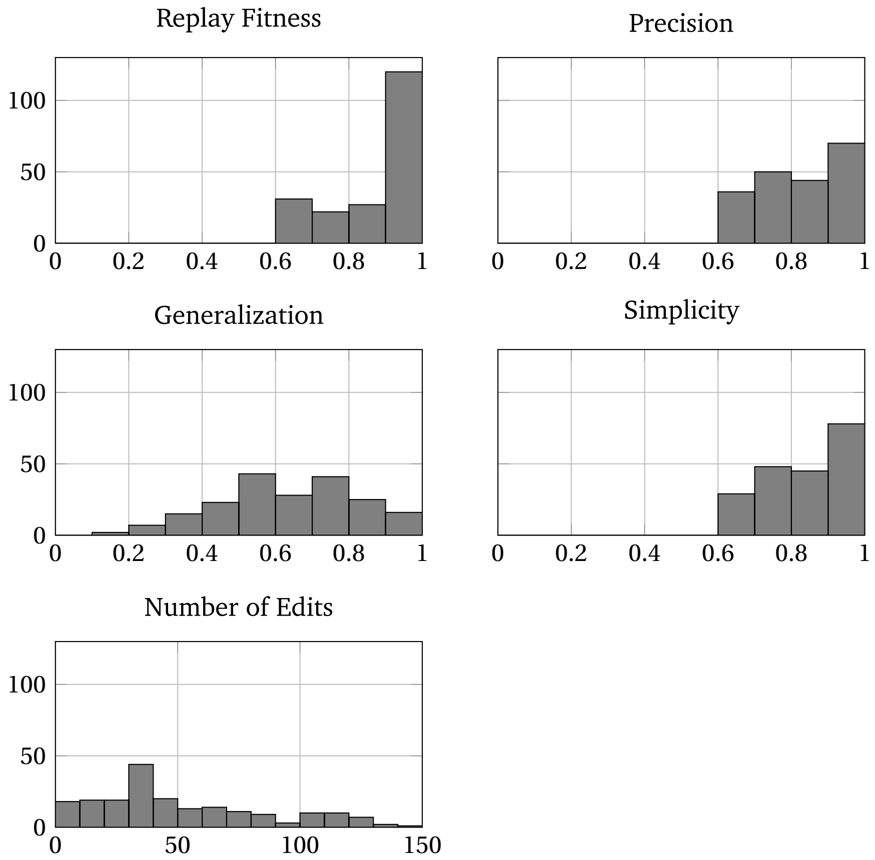


Figure 8.9: Distribution over the quality dimensions of 200 candidates in the Pareto front for the case study.

precision and simplicity. The distribution of the process trees over the values for the different quality dimensions is shown in Figure 8.9. This demonstrates that most process trees have high replay fitness. At most 141 edits have been applied to a process tree, which results in a process tree with little resemblance to the reference process model. Relatively more process trees have between 30 and 40 edits applied than other edit ranges.

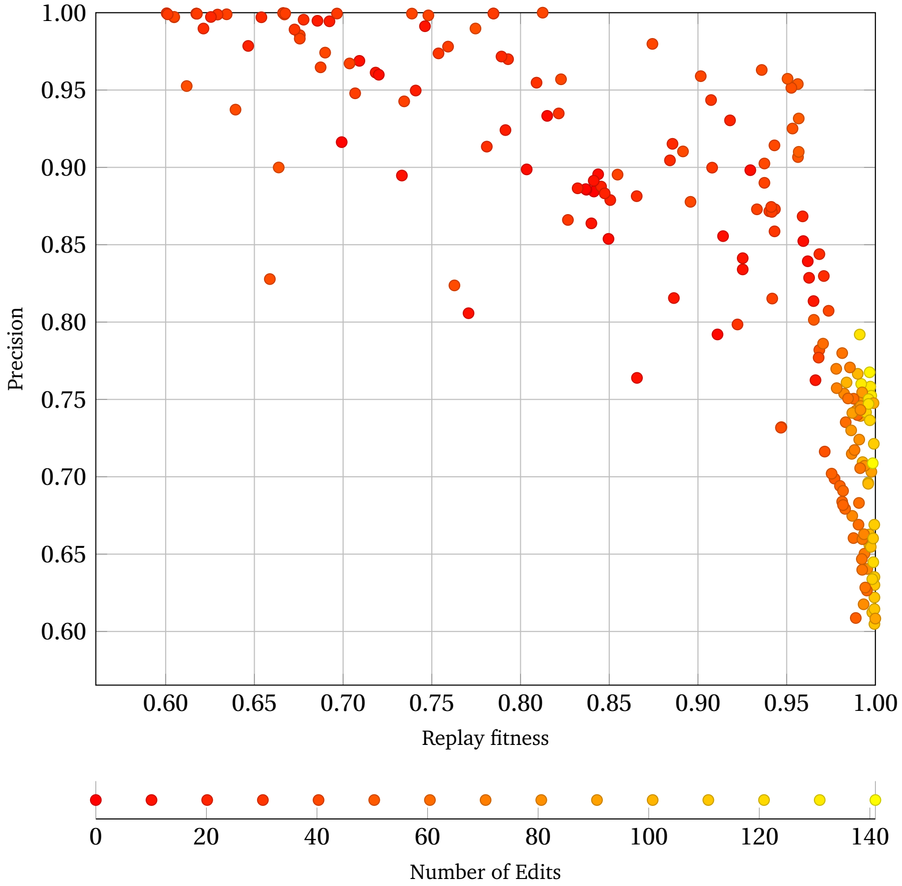


Figure 8.10: Visualization of the Pareto front of process trees discovered by the ETM*r* algorithm on the case study event log.

The Pareto front projected on the two dimensions of replay fitness and precision is shown in Figure 8.10, where the color indicates the number of edits applied. This visualization shows that balancing replay fitness and precision appears to be difficult for this event log.

Figure 8.11 shows the different Pareto fronts for different edit ranges. This demonstrates that, especially for up to 50 edits, the Pareto front moves towards both better replay fitness and better precision. Process trees with 50 or more edits applied seem to mainly improve on replay fitness, instead of precision.

A selection of process trees from the Pareto front is shown in Figure 8.12. When allowing at most 10 edits, as is shown in the filtered Pareto front at the top-left of Figure 8.11, less improvements can be made. A process tree with at most 10 edits, that provides a good tradeoff between replay fitness and precision, is shown in Figure 8.12a. This process model is mainly improved by adding activity a (T02 Check confirmation of receipt) next to activity l (Confirmation of receipt) and removing a big part of the 'redo' part of the leftmost ↺-operator, all of which was done in 7 edits. Applying these edits improved replay fitness from 0.841 to 0.930 and improved precision from 0.884 to 0.898. The nodes on the left-hand side of the model, in the loop with e, for instance remain after removing parts of the process tree and become the two useless nodes of the process model. When removing these two useless nodes the resulting process model shows many similarities with the normative process model, while replay fitness and precision are significantly improved.

The process tree with the best replay fitness given perfect precision is shown in Figure 8.12b. By applying 39 edits the process model is reduced in size making it more precise and significantly more general, at the cost of replay fitness. A good trade-off between replay fitness and precision can be obtained within 49 edits of the normative process model, as is shown in Figure 8.12c. This process tree also demonstrates that useless nodes are not removed since it might increase the edit distance. Obtaining a perfect replay fitness score requires 92 edits on the normative model, resulting in the process tree shown in Figure 8.12d. These process trees demonstrate that the normative model does not describe all observed behavior and requires quite some modification for it to do so.

This case study demonstrates that the Pareto front of (repaired) process trees discovered by the ETM*r* algorithm provides several insights into the relationship between the modeled and observed process. By providing process models with different gradations of change applied, the municipality is able to investigate how the process as they know it is actually executed. For instance, it became clear that by applying seven edits, the process model can be significantly repaired based on the observed behavior. Many more repairs, up to 150, can be performed to improve the process model to better reflect the observed behavior. Moreover, the Pareto front, by applying several filters, provides a way to investigate several repaired process models.

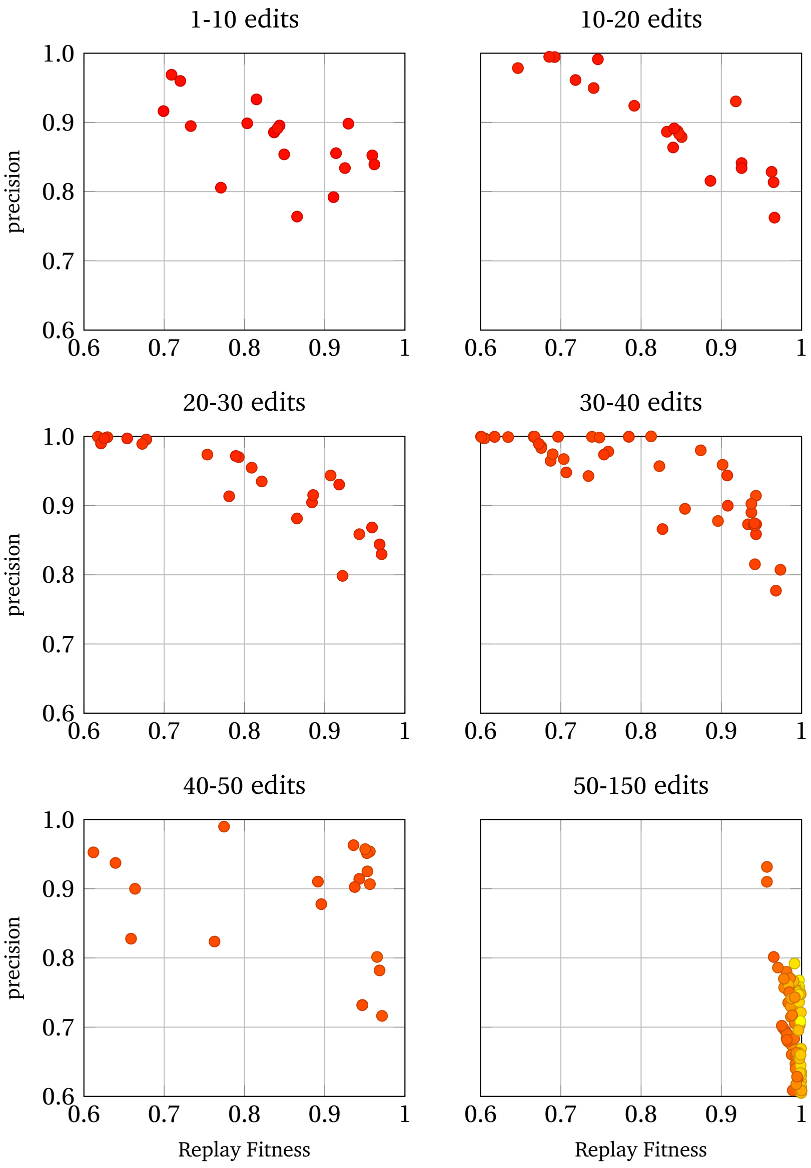
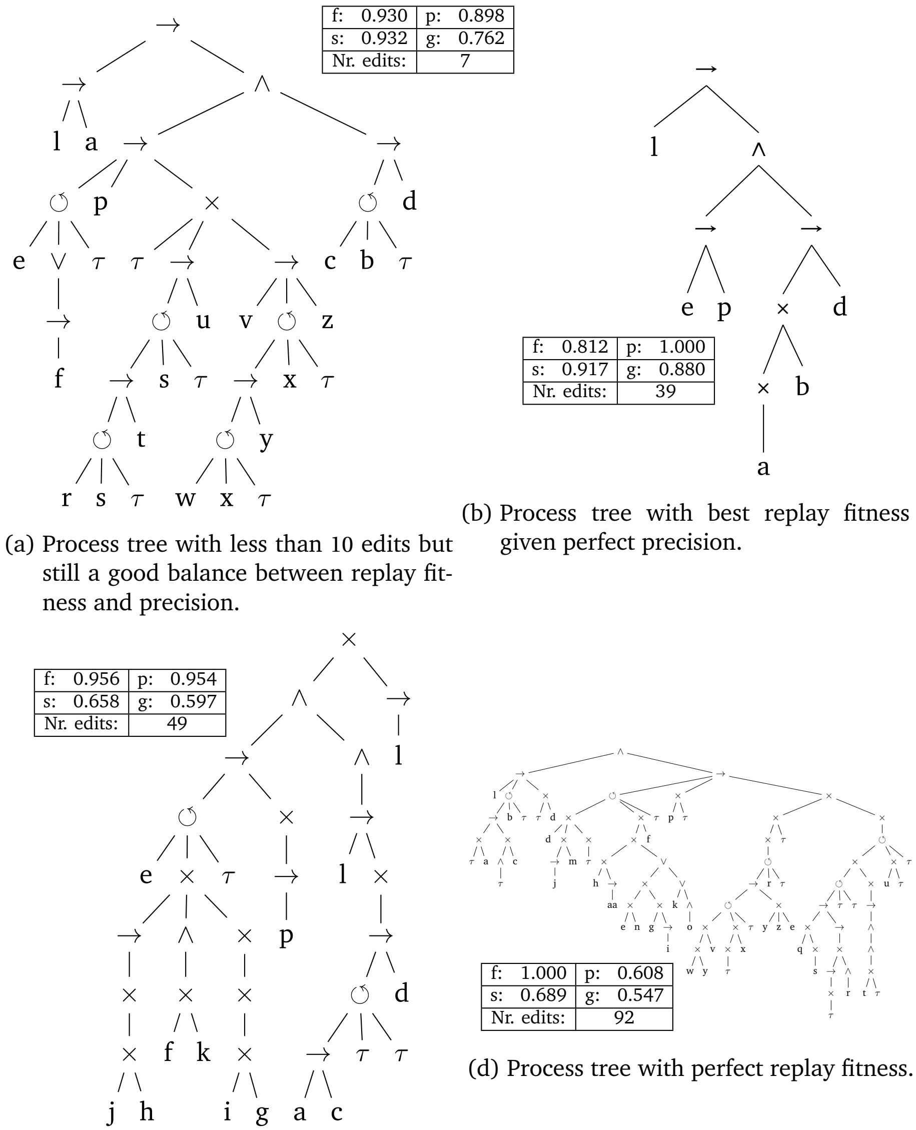


Figure 8.11: Visualization of the Pareto front filtered by the number of edits allowed.



f: 0.930	p: 0.898
s: 0.932	g: 0.762
Nr. edits:	7

(a) Process tree with less than 10 edits but still a good balance between replay fitness and precision.

f: 0.812	p: 1.000
s: 0.917	g: 0.880
Nr. edits:	39

(b) Process tree with best replay fitness given perfect precision.

f: 0.956	p: 0.954
s: 0.658	g: 0.597
Nr. edits:	49

(c) Process tree with a good trade-off between replay fitness and precision.

f: 1.000	p: 0.608
s: 0.689	g: 0.547
Nr. edits:	92

(d) Process tree with perfect replay fitness.

Figure 8.12: Different process trees selected from the Pareto front discovered by the ETM*r* algorithm on the case study.

8.5 Related Work

Only a few approaches exist that consider similarity in the context of process discovery. Several approaches exist that consider similarity between process models, without the use of an event log.

Li et al. [124–126] discuss how a reference process model can be discovered from a collection of process model variants. In their heuristic approach they consider the structural distance of the discovered reference model to the original reference model as well as the structural distance to the process variants. By balancing these two forces they make certain changes to the original reference model to make it more similar to the collection of process model variants. Compared to our approach, here the starting point is a collection of process variants, rather than an event log.

An approach aiming to automatically correct errors in an *unsound* process model (i.e., a process model affected by behavioral anomalies) is presented by Gambini et al. [86]. Their approach considers three dimensions: the structural distance, behavioral distance and 'badness' of a solution with respect to the unsound process model. The 'badness' dimension indicates the ability of a solution to produce traces that lead to unsound behavior. The approach uses simulated annealing to simultaneously minimize all three dimensions. The edits applied to the process model are aimed at correcting the model rather than balancing the four quality dimensions used in process discovery.

Within the area of process mining there currently is a gap between process discovery and conformance analysis. Conformance analysis, such as the work of Adriansyah et al. [21, 24, 25], detects conformance of, and deviations from, observed behavior compared to modeled behavior. When deviations are detected, these are visualized on either the process model or the observed behavior. However, no fixes are suggested to improve the process model using the observed behavior. Vice versa, only a few process discovery algorithms exist that are able to incorporate a given process model and improve or repair it.

The work of Fahland et al. [79, 80] provides a first integrated attempt at repairing process models based on alignments between the process model and event log. In their approach, a process model needs repair if the observed behavior cannot be replayed by the process model. This is detected by using the alignment between the process model and the observed behavior. The detected deviations are then repaired by extending the process model with sub-processes nested in a loop block. These fixes are applied repeatedly until a process model is obtained that can perfectly replay the observed behavior. This approach extends the original process model's behavior by adding new fragments that enable the model to replay the observed behavior. As a post-processing step some infrequently used parts of the process model can be removed to improve precision. The main disadvantage of this approach is that mainly the replay fitness quality dimension is considered when repairing mis-alignments. Although post-processing steps are introduced to improve precision, these steps do not always succeed. Moreover, since repairs mainly add transitions to the model, by definition the model can only become more complex. It is unclear how to correctly balance all five quality dimensions when extending the work in [79, 80].

8.6 Conclusion

In this chapter we presented the ETM*r* algorithm which extends the ETM*d* algorithm to consider a given process model. The main goal is to mediate between the observed behavior and some a priori model (e.g. a reference model). The desired trade-off is achieved by adding an additional quality dimension that evaluates the edit distance between the provided process model and the discovered process model. We applied the ETM*r* algorithm on both a running example and a real life event log. By constructing a Pareto front, the amount of change applied to the process model can be varied to investigate how much change should be applied to better describe the observed behavior.

Chapter 9

Discovering Configurable Process Models

In this chapter we address Challenge 6: "Describe a family of processes". Different organizations or units within a larger organization often execute similar business processes. Municipalities for instance all provide similar services while being bound by government regulations. Large car rental companies like Hertz, Avis and Sixt have offices in different cities and airports all over the globe. Often there are subtle (but sometimes also striking) differences between the processes handled by these offices, even though they belong to the same car rental company. To be able to share development efforts, analyze differences, and learn best practices across organizations, we need *configurable process models* that are able to describe a family of process variants rather than one specific process [93, 116, 154].

Given a collection of event logs that describe similar behavior we can discover a process model using existing process mining techniques [5]. However, existing techniques are not tailored towards the discovery of a *configurable* process model based on a *collection* of event logs. In this chapter, we present and compare four approaches to mine configurable models. The first two ap-

Parts of this chapter are based on the work as presented in [51]:
J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Mining configurable process models from collections of event logs. In F. Daniel, J. Wang, and B. Weber, editors, *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40175-6.

proaches use a combination of existing process discovery and process merging techniques. The third approach uses a two-phase approach where first a process tree is discovered which is then configured. The fourth approach uses a new, integrated approach, implemented in the ETM*c* algorithm.

9.1 Configurable Process Models

A configurable process model describes a *family* of process models, i.e., variants of the same process. A configurable process model contains additional options to configure, or individualize, the process model to one of the process model variants. Two different mechanisms can be used: restriction and extension of behavior. A configurable process model notation can restrict the behavior by removing activities or activity combinations. The second mechanism is to allow extensions of the configurable process model, i.e., adding behavior, to obtain the process model variant. Some configurable process modeling notations support both mechanisms, and all notations allow for restricting the behavior [116]. In this section we briefly discuss three configurable process model notations: C-EPCs, C-YAWL and PROVOP, which extends BPMN. A more extensive discussion on configurable process model notations can be found in [116].

Configurable EPCs (C-EPCs) [70,71,117,154] extend the EPC language (see Section 3.2.3). An example C-EPC is shown in Figure 9.1. C-EPCs restrict the described behavior by applying configuration options. In C-EPCs configurable control-flow connectors can be configured to be an equally or more restrictive connector. If a function (i.e., activity) is made configurable then this function can be set as activated (on), deactivated (off) or allowed to be skipped. In the example of Figure 9.1 for instance, functions `a`, `e` and `f` are configurable, as is indicated by the thicker borders. Furthermore, the XOR connector is also configurable and allows one or more of its outgoing paths to be blocked. Additionally, requirements can be specified which restrict the possible configurations allowed. Guidelines can also be specified which suggest configuration settings, for instance that if either activity `e` or `f` is on, the other should also be on.

The C-YAWL notation [14,93], which is an extension of the YAWL notation (see Section 3.2.2), allows YAWL models to be configured. Process models can be customized by applying two operators to process model activities: *hiding* and *blocking*. Both operators restrict the behavior of the process model and are applied on the ports of activities. A port is a combination of allowed incoming or outgoing edges for a particular activity. Hiding results in the execution of that activity to become unobservable, but the path to the activity is in is still possible. Blocking disables the execution of the activity, which means that the path the activity is in cannot be taken anymore. An example of a C-YAWL model with two configurations applied is shown in Figure 9.2. A green arrow indicates that no restriction is configured. Hiding is indicated by an orange arrow, while blocking is indicated by a stop-sign. In the example of Figure 9.2 the process model variant for the travel agency only allows the activity `book reduction`

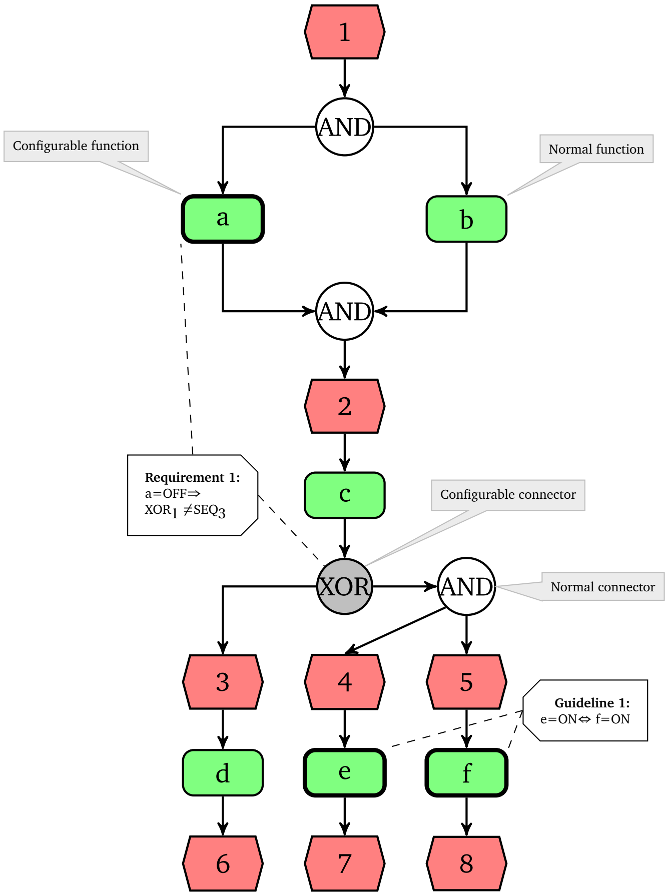


Figure 9.1: Example of a configurable process model in C-EPC notation (adapted from [154]).

`card` to be executed if `book train ticket` is also enabled. This is indicated by blocking the outgoing ports `b` and `b,c` of activity `Receive order`, while the outgoing port `a,b` is not blocked. For the internet shop the activity `select payment method` can be skipped since only credit card payments are accepted.

Several extensions to the BPMN notation exist that introduce configuration options [98, 112, 113, 140, 160]. Figure 9.3 shows an example of one of these approaches, the PROVOP (PROcess Variants by OPtions) approach [98]. Within the PROVOP approach four operations are defined: delete, insert, move and modify. Applying the delete operation removes part of the process model, and thus restricts the allowed behavior. Insertion extends the behavior of the process model by adding predefined process model fragments at specified locations. The move operation allows a specific process model part to move to one of several other predefined locations in the process model. Finally, the modify operation allows attributes of model elements to change, such as the role assigned to an activity.

Restricting behavior is common across all configurable process modeling notations. However, only the C-EPC notation allows for operator downgrading, which also limits behavior. In the next section we propose to extend process trees by allowing for the restriction of the behavior in two ways: by removing parts of the process tree and by downgrading operators.

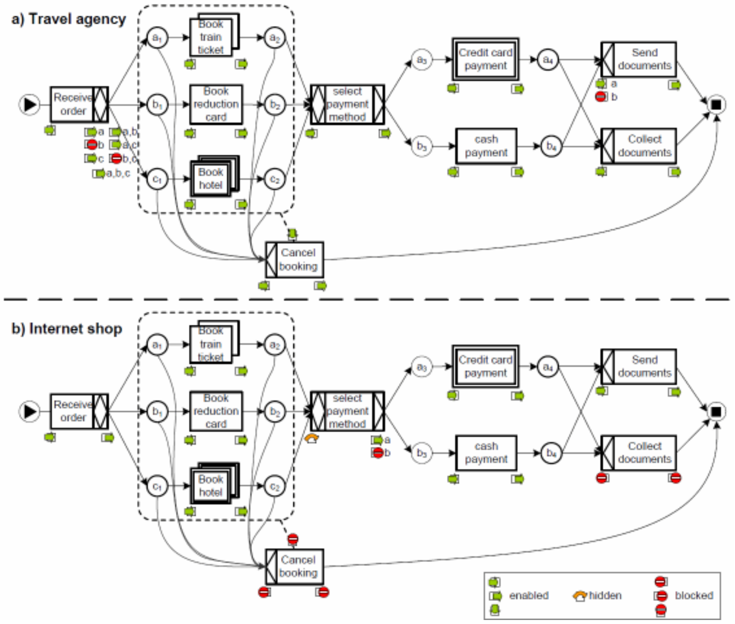


Figure 9.2: Examples of configurable process models in C-YAWL notation (from [93]).

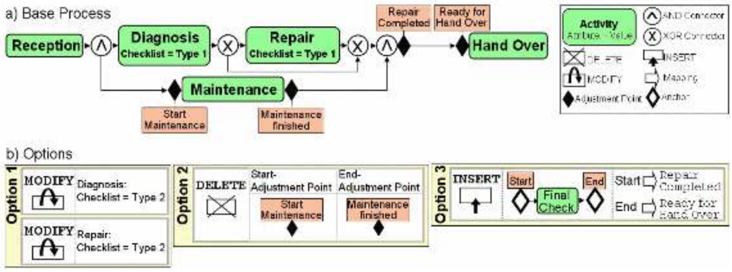


Figure 9.3: Example of a configurable process model in PROVOP notation (from [98]).

9.2 Configurable Process Trees

In this section we extend the process tree notation to configurable process trees such that a configurable process tree describes a family of process trees. By applying a *configuration*, an individualized process tree is obtained. Each node in the configurable process tree contains a *configuration point* for each configuration. Each configuration point can be set to a *configuration option*, which can be *blocked* or *hidden*, and operator nodes can be *downgraded* to another operator type. By applying all configurations options for all configuration points of a specific configuration to a configurable process tree, an individual process tree is obtained.

Hiding and blocking work in a similar way as in the C-YAWL notation but on nodes instead of ports. Hiding makes the node unobservable and in essence replaces it with a τ-node. If a node is blocked, the path leading to that activity cannot be taken anymore. In case of blocking, several situations can be distinguished, which are visually explained in Figure 9.4:

1. In case the parent of the blocked node is an $\times$- or $\vee$-operator:

   (a) If the blocked node is the last remaining child, then the parent itself is also blocked (Figure 9.4b and Figure 9.4d).

   (b) Otherwise, the blocked node is removed from the parent (Figure 9.4a and Figure 9.4c).

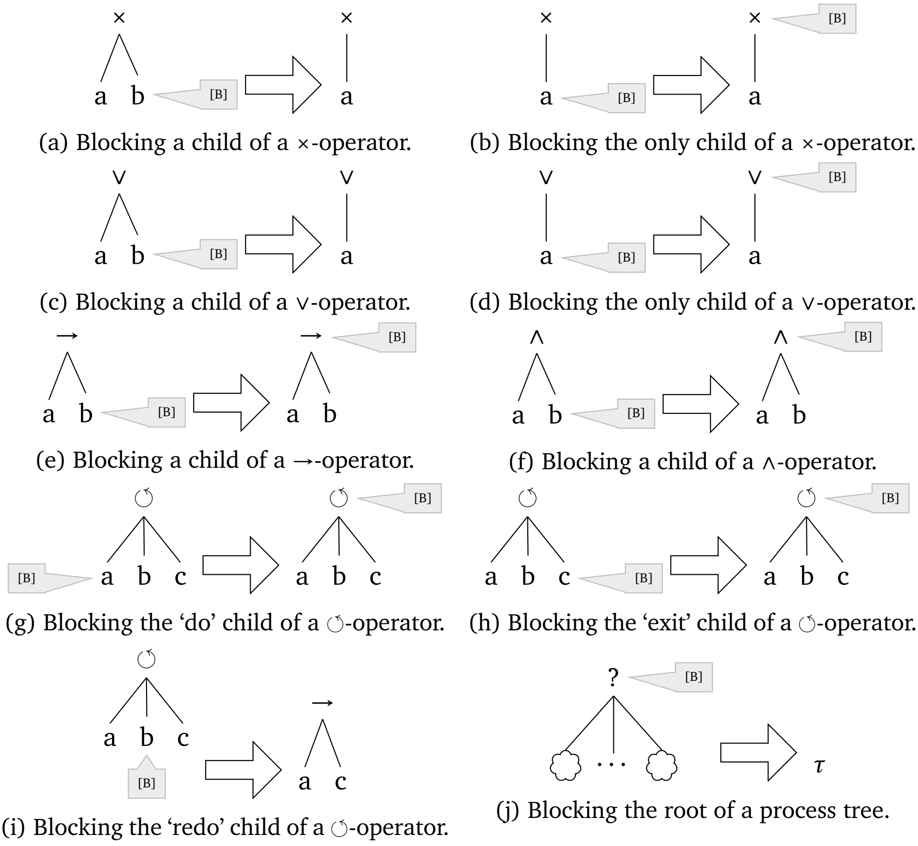
2. In case the parent of the blocked node is an $\rightarrow$- or $\wedge$-operator, that operator is also blocked, since it enforces the execution of all of its children (see Figure 9.4e and Figure 9.4f).

3. In case the parent of the blocked node is an $\circlearrowleft$-operator:

   (a) If the blocked node is the 'do' or 'exit' child, then the parent is also blocked (see Figure 9.4g and Figure 9.4h).

   (b) If the blocked node is the 'redo' child, then the $\circlearrowleft$ parent is replaced by an $\rightarrow$-operator with the 'do' and 'exit' of the loop node as children (see Figure 9.4i).

4. In case the blocked node is the root node then the root is replaced by a τ-node since the process tree does not allow for any behavior (see Figure 9.4j).



(a) Blocking a child of a ×-operator.

(b) Blocking the only child of a ×-operator.

(c) Blocking a child of a ∨-operator.

(d) Blocking the only child of a ∨-operator.

(e) Blocking a child of a →-operator.

(f) Blocking a child of a ∧-operator.

(g) Blocking the ‘do’ child of a ↺-operator.

(h) Blocking the ‘exit’ child of a ↺-operator.

(i) Blocking the ‘redo’ child of a ↺-operator.

(j) Blocking the root of a process tree.

Figure 9.4: Effects of blocking nodes in a process tree

Since blocking a node can have an effect higher up in the process tree, the configurations have to be applied recursively.

Additionally we allow for operators to be *downgraded* following the downgrade hierarchy shown in Figure 9.5. By downgrading an operator, the behavior of the operator is restricted to a subset of the initially possible behavior. The ∨-operator for instance can be downgraded to an ∧- (forcing all children to be executed), ×- (only allowing for one child to be executed), and a →-operator (executing all children in a particular order). However, since in one configuration the order of the children might be different than in another, the ←-operator, representing a *reversed sequence*, is added which executes the children in the reversed order, i.e., from right to left. Similar to the ∨, an ∧ can be downgraded to an →- or ←-operator.

An example of operator downgrading is shown in Figure 9.6 where an ∨-operator is downgraded to a →-operator. Configuration points are visualized using a gray callout, pointing at the node to which it belongs. Within the callout the configuration options set for each configuration are displayed in a sequence. The process tree of Figure 9.6 contains one configuration point and one configuration, set to the →-operator.

9.3 Four Different Approaches

We consider four approaches to discover a configurable process tree from a collection of event logs. These four approaches are shown in Figure 9.7.

Approach 1, as is shown in Figure 9.7a, applies process discovery on each input event log to obtain the corresponding process model. Then these processes models are merged using model merge techniques. This approach was first proposed in [92]. We execute this approach by applying the ETM*d* on each of the

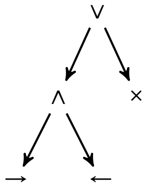


Figure 9.5: Hierarchy of operator downgrade options.

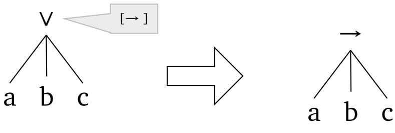
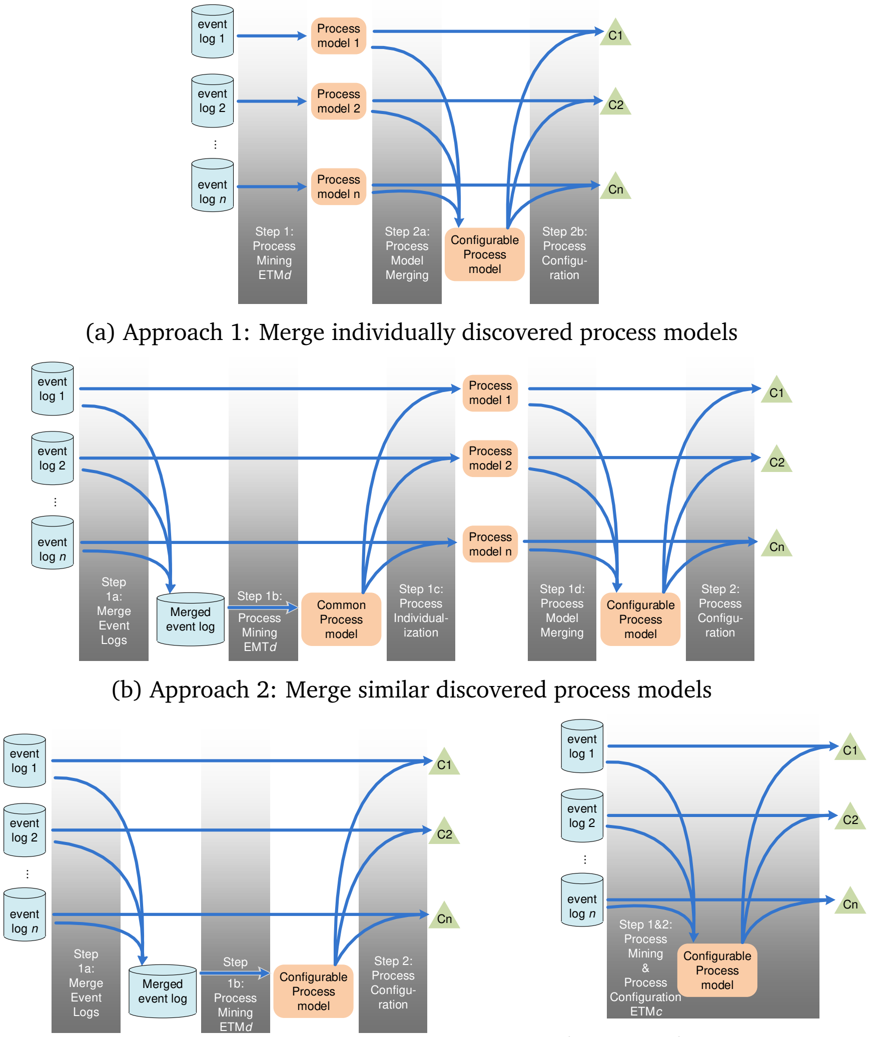


Figure 9.6: Example of downgrading an ∨-operator to an →-operator.


event log 1
event log 2
event log *n*
Process model 1
Process model 2
Process model n
C1
C2
Cn
Step 1: Process Mining ETM*d*
Step 2a: Process Model Merging
Configurable Process model
Step 2b: Process Configu-ration
Step 1a: Merge Event Logs
Merged event log
Step 1b: Process Mining EMT*d*
Common Process model
Step 1c: Process Individual-ization
Step 1d: Process Model Merging
Step 2: Process Configu-ration
Step 1b: Process Mining ETM*d*
Step 1&2: Process Mining & Process Configuration ETM*c*

(a) Approach 1: Merge individually discovered process models

(b) Approach 2: Merge similar discovered process models

(c) Approach 3: First discover a single process model and then discover configurations

(d) Approach 4: Discover process model and configurations at the same time

Figure 9.7: Four approaches to creating a configurable process model from a collection of event logs.

input event logs and merging the resulting process trees using the technique presented in [161].

Since the process models of the first approach are discovered independently of each other, they might differ significantly, hence merging them correctly is difficult. Therefore we present Approach 2 as an improvement of the previous approach. The overall idea is shown in Figure 9.7b. From the merged input event logs first one process model is discovered that describes the behavior recorded in all event logs. Then the single process model is taken and individualized for each event log using the ETM*r* algorithm as discussed in Chapter 8. In the next step these individual process models are merged into a configurable process model using the approach of [161]. By making the individual process models more similar, merging them into a configurable process model should be easier.

Approach 3, as shown in Figure 9.7c, is an extension of the second approach presented in [92]. A single process model is discovered that describes the behavior of all event logs. Then, using each individual event log, configurations are discovered for this single process model. In this approach the common process model should be less precise than in the other approaches since we can only restrict the behavior using configurations, but not extend it. Therefore, the ETM*d* algorithm applied needs to put less emphasis on precision. For the second phase we only change the configuration options without changing the structure of the process tree. We do this by applying the ETM*c* algorithm, which we introduce in Section 9.4, without change operations that modify the tree structure.

The fourth approach is a novel approach implemented in the ETM*c* algorithm where the discovery of the process model and the configurations is combined, see Figure 9.7d. This approach is added to overcome the disadvantages of the other three approaches. By providing an integrated algorithm, where both the process model and the configuration options are discovered simultaneously, better trade-offs between the different quality dimensions can be made. In the next section we present the ETM*c* algorithm.

9.4 The ETM*c* algorithm

The ETM*c* algorithm is an extension of the ETM*d* algorithm for discovering configurable process trees. The input of the ETM*c* algorithm is a collection of event logs and the output is a configurable process model. For each of the input event logs a configuration is discovered that blocks, hides, or downgrades certain nodes in the configurable process tree. Besides operating on configurable process trees, two elements are added to the ETM*d* algorithm: changing the configurations and evaluating the configurable process tree.

9.4.1 Configuration Mutation

Initially a process tree has one configuration for each input event log, where none of the configuration points is configured. Currently we apply a random mutation to the configurable process tree that picks a configuration for one of the event logs. It then randomly selects a configuration point and randomly changes the configuration option to one of the allowed options for that configuration point. This simple random mutation allows coverage of the whole search space.

9.4.2 Configuration Quality

The overall quality of the configurable process tree consists of two aspects: the quality of each of the individualized process trees (i.e., the process trees obtained after application of the configurations), and the quality of the configuration aspect of the configurable process tree. These two aspects are weighted using an α parameter. The overall quality of a configurable process tree is thus evaluated as follows:

Definition 9.1 *(Quality of a configurable process tree)*
Let LC be the collection of all input event logs such that $L \in LC$ is an event log in this collection. Let PT^c be a configurable process tree and PT^c_L the individual process tree obtained from PT^c for event log L. Furthermore, let the function $Q(PT^c_L, L)$ be a quality metric for PT^c_L on the corresponding event log L.

We define the average quality in a certain quality dimension as follows:

$$\overline{Q}(PT^c, LC) = \frac{\sum_{L \in LC} |L| \times Q(PT^c_L, L)}{\sum_{L \in LC} |L|} \tag{9.1}$$

We define the configuration quality as follows:

$$Q_c(PT^c) = 1 - \frac{\textit{number of configured nodes in } PT^c}{\textit{number of nodes in } PT^c} \tag{9.2}$$

Table 9.1: Four event logs for the four different variants of the loan application process of Section 1.2.2.

(a) Event log for variant 1

Trace	#
a b c d f g	38
a b d c f g	26
a b d c e g	12
a b c f g	8
a b c d e g	6
a d c b f g	4
a c d b f g	2
a c b e g	1
a d b c f g	1
a d b c e g	1
a c b f g	1

(b) Event log for variant 2

Trace	#
a b1 b2 c d2 f	50
a b1 b2 c d2 e	20

(c) Event log for variant 3

Trace	#
a c b e	120
a c b f	80

(d) Event log for variant 4

Trace	#
a b1 d2 b2 c f	60
a b1 d b2 c e	45

9.5 Application on Running Example

Our running example is based on four variants of the same process. The event logs of this running example are shown in Table 9.1. The four variants are discussed in more detail in Section 1.2.2.

Although the four variations of the loan application process seem similar, automatically discovering a configurable process model turns out to be far from trivial.

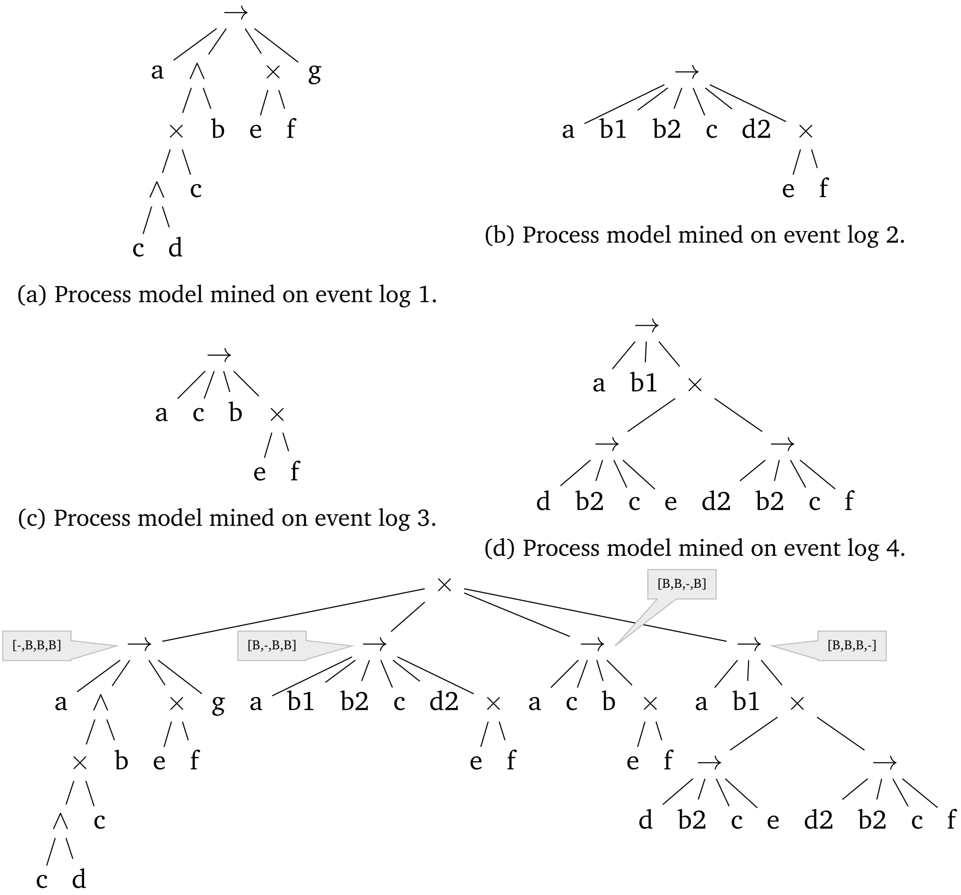
9.5.1 Experimental Setup

In the remainder of this section we use the ETM*d* and ETM*c* algorithms as our discovery techniques to construct a process model, in the form of a process tree, from an event log. For Approach 1 the ETM*d* algorithm is ran for 10,000 generations on each event log, after which the resulting process models were merged. For Approach 2 the ETM*d* algorithm first ran for 5,000 generations on the merged event log of the four variants. This was followed by 10,000 generations per event log of the ETM*r* algorithm in order to individualize the process models before merging. In Approach 3 the ETM*d* algorithm ran for 10,000 generations to discover an imprecise process model for the combined event log, after which configurations were discovered using the ETM*c* algorithm during another 10,000 generations while retaining the control-flow structure. Within Approach 4 the ETM*c* algorithm ran for 10,000 generations discovering the process tree structure and configurations at the same time. The quality dimension of replay fitness is given a weight of ten, and a weight of five for precision made sure the model does not allow for too much additional behavior. Simplicity is weighted by one, and a weight of one-tenth for generalization makes the models more general. In Approach 2 the similarity quality dimension, as discussed in Section 8.2, is added with a weight of five. The rest of the settings are the same as used in the experiments as discussed in Chapter 7.

For the resulting configurable process trees we calculate the four quality dimensions for each of the individualized process trees on the corresponding event log. We aggregate these values to the configurable process tree by using a weighted average using the number of traces in each event log. Additionally, the size of both the configurable process tree and the individualized process trees are calculated. The number of configuration points (#C.P.) set in the configurable process tree, as well as the number of configuration points applied per variant, are shown in a table for each approach. The similarity of individual process trees to the configurable process tree is also calculated.

9.5.2 Approach 1: Merge Individually Discovered Process Models

The results of applying Approach 1 on the running example are shown in Figure 9.8. Each of the individual process models (see Figure 9.8a through Figure 9.8d) clearly resembles the corresponding event log. The combined configurable process model as shown in Figure 9.8e however is nothing more than a choice between each of the individual input process models. The table shown in Figure 9.8f shows the different quality scores for both the configurable process models and for each of the configurations. Moreover, the simplicity statistics of size, number of configuration points (#C.P.) and similarity of the configured process model to the configurable process model are shown. The fact that the four configuration options block a big part of the process model is reflected in the low similarity of the configured process models with the configurable process model. This is also shown by the relatively large size of the configurable process tree with respect to the size of the individual process tree variants.



(a) Process model mined on event log 1.

(b) Process model mined on event log 2.

(c) Process model mined on event log 3.

(d) Process model mined on event log 4.

(e) Configurable process tree.

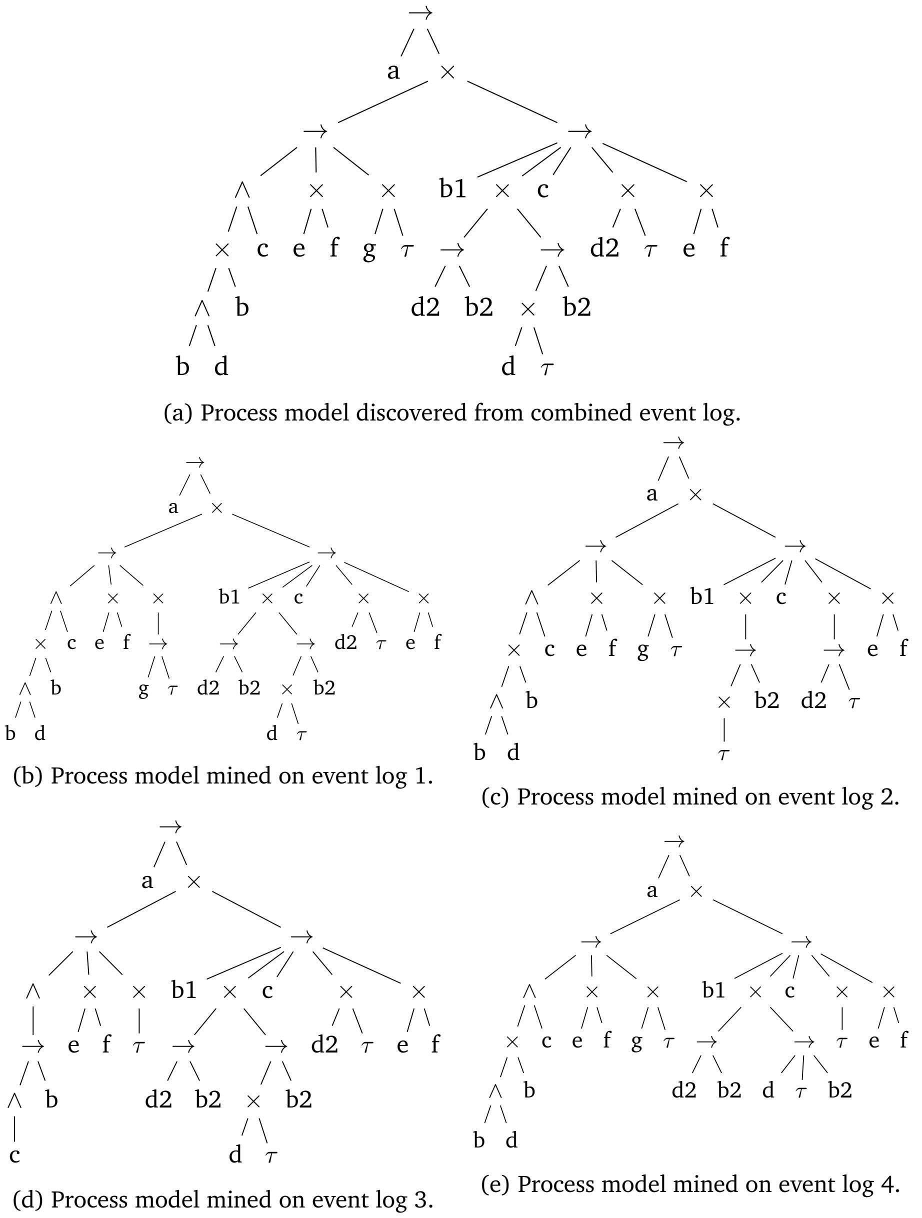
(f) Quality statistics of the configurable process model of (e).

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.975	1.000	0.998	0.903	0.805	44	4	-
Variant 1	0.982	1.000	0.990	0.929	0.809	14	3	0.483
Variant 2	0.981	1.000	1.000	0.900	0.780	10	3	0.370
Variant 3	0.981	1.000	1.000	0.875	0.805	8	3	0.308
Variant 4	0.985	1.000	1.000	0.933	0.815	15	3	0.508

Figure 9.8: Results of Approach 1, merging separately discovered process models, on the running example.

9.5.3 Approach 2: Merge Similar Discovered Process Models

In Approach 2 we try to increase similarity amongst the individual process models by discovering a common process model from all event logs combined. This combined model for the input event logs is shown in Figure 9.9a. This process model has difficulties with describing the combined behavior of the four variants. For instance, it tries to distinguish variants 1 and 3 from variants 2 and 4 by introducing an ×-operator high in the process tree. The four individual process models derived from this common process model are shown in Figure 9.9b through Figure 9.9e. However, the unused child of the ×-operator for a variant is not removed, since it would significantly decrease similarity with the combined model. This results in each of the individual process trees being large and having relatively low scores for precision and generalization. The four individual process trees are however very similar to each other. The combined process tree is shown in Figure 9.10a. Despite the similarity of the individual process models, the combined configurable process model is still a choice between the four input process models. Figure 9.10b shows the statistics of this configurable process model. The overall quality of this model is worse than that of Approach 1, which is mainly due to the lower scores for precision and generalization. Similar to the previous approach, the number of configuration points is low. Unfortunately, the similarity between the configurable process model and the process model variants is also low.



(a) Process model discovered from combined event log.

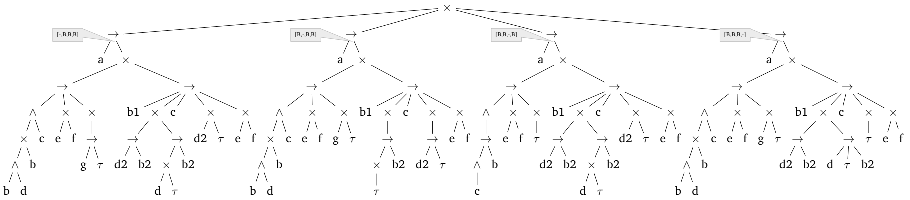
(b) Process model mined on event log 1.

(c) Process model mined on event log 2.

(d) Process model mined on event log 3.

(e) Process model mined on event log 4.

Figure 9.9: Individual results of approach 2, merging the similar process models, on the running example.


×
[-,B,B,B]
→
[B,-,B,B]
→
[B,B,-,B]
→
[B,B,B,-]
→
a × a × a × a ×
→ → → → → → → →
∧ × × b1 × c × × ∧ × × b1 × c × × ∧ × × b1 × c × × ∧ × × b1 × c × ×
× c e f → → → d2 τ e f × c e f g τ → → e f → e f τ → → d2 τ e f × c e f g τ → → τ e f
∧ b g τ d2 b2 × b2 ∧ b × b2 d2 τ ∧ b d2 b2 × b2 ∧ b d2 b2 d τ b2
b d d τ b d τ c d τ b d

(a) Configurable process tree.

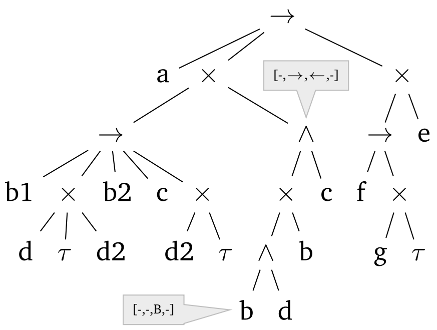
(b) Quality statistics of the configurable process model of (a).

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.929	1.000	0.891	0.890	0.362	134	4	-
Variant 1	0.937	1.000	0.926	0.919	0.379	37	3	0.433
Variant 2	0.927	1.000	0.909	0.848	0.370	33	3	0.395
Variant 3	0.919	1.000	0.889	0.879	0.308	33	3	0.395
Variant 4	0.918	1.000	0.851	0.912	0.441	34	3	0.405

Figure 9.10: Configurable process model with quality statistics as discovered by Approach 2, merging the similar process models, on the running example.

9.5.4 Approach 3: First Discover a Single Process Model and Then Discover Configurations

In Approach 3 we first try to discover a process tree that describes the combination of all event logs. This is achieved by reducing the weight for precision by a factor 10, resulting in a weight of 0.5 for precision. In the second phase we only change configuration options to increase the precision of configured process trees, but the structure of the configurable process tree remains the same. The resulting configurable process model is shown in Figure 9.11. From this model it can be seen that we relaxed the precision weight, in order to discover an ‘over fitting’ process model. Then, by applying configurations, the behavior is restricted in such a way that the model more precisely describes each of the variants, as is indicated by the perfect replay fitness for all but variant 1. The resulting configurable process model however scores lower on precision the previous two approaches. This might be caused by a disconnect between the two phases. It appears that adding configuration options in the second phase



(a) Configurable process tree.

(b) Quality statistics of the configurable process model of (a).

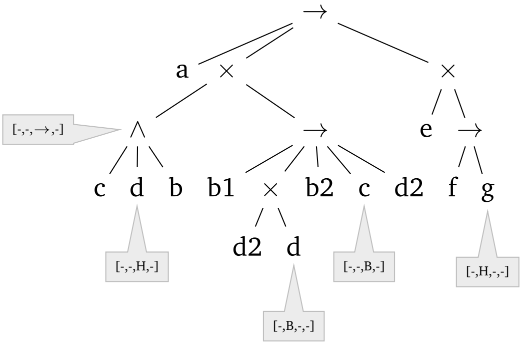
	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.908	0.994	0.794	0.983	0.494	28	2	-
Variant 1	0.920	0.973	0.880	1.000	0.500	28	0	1.000
Variant 2	0.882	1.000	0.694	1.000	0.528	28	1	0.982
Variant 3	0.918	1.000	0.841	0.960	0.439	25	2	0.925
Variant 4	0.883	1.000	0.689	1.000	0.571	28	0	1.000

Figure 9.11: Results of Approach 3, the two-phase mining approach, on the running example.

cannot restrict the behavior of the process model sufficiently. The resulting configurable process tree however is much smaller and has less configuration points than the results of Approaches 1 and 2. Moreover, the similarity of each of the individualized process trees to the configurable process tree is high.

9.5.5 Approach 4: Discover Process Model and Configurations at the Same Time

The result of applying Approach 4, an integrated approach, is shown in Figure 9.12. This configurable process tree is smaller than the ones obtained by the first three approaches. Moreover, it clearly includes the common parts of all variants only once, e.g. it always starts with a and ends with a choice between f, which is sometimes followed by g, and e. This process model hides some of the activities that do not occur in certain variants, for instance activity d for variant 2. However, it does not find all configuration options, since g can also



(a) Configurable process tree.

(b) Quality statistics of the configurable process model of (e).

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.913	0.957	0.934	0.946	0.646	20	5	-
Variant 1	0.931	0.970	0.919	1.000	0.532	20	0	1.000
Variant 2	0.913	0.950	0.915	0.895	0.548	19	2	0.949
Variant 3	0.963	0.971	1.000	0.909	0.751	11	3	0.677
Variant 4	0.883	0.921	0.835	1.000	0.622	20	0	1.000

Figure 9.12: Results of Approach 4, the integrated mining approach, on the running example.

be hidden for variants 3 and 4 without reducing replay fitness. In this approach the parallelism present in variant 1 is correctly discovered (in the left-most subtree) and downgraded for variant 3. Variants 2 and 4 are explained by the right branch of the leftmost ×-operator under the root. However, the two children of the ×-operator are not correctly blocked to increase precision for each of the variants.

9.5.6 Discovering a Pareto Front for Approach 4

The ETM*c* algorithm can also construct a Pareto front of candidates, where the number of configuration points is used in addition to the other four quality dimensions. The other four quality dimensions of replay fitness, precision, generalization and simplicity are aggregated over the individual values for the different configurations and weighted by the number of traces in the event log.

The ETM*c* algorithm discovered a Pareto front containing 382 candidates for the running example. The distribution of the candidates over the five quality dimensions is shown in Figure 9.13. The Pareto front projected on the quality dimensions of replay fitness and precision, where the color indicates the number of configuration points, is shown in Figure 9.14. This shows that the best trade-off between replay fitness and precision is made with two or three configuration points. Two of the process trees with three configuration points are shown in Figure 9.15 and Figure 9.16. They demonstrate slightly different trade-offs between replay fitness and precision. The process tree of Figure 9.15 is reasonably precise, while scoring good on replay fitness. This is achieved mainly by introducing a choice between three different descriptions of the middle part of the process. Configuration options are added only for the third variant. Figure 9.16 shows a process tree that scores very well on replay fitness, but very badly on precision. This is mainly caused by the ∧-operator as root, which is downgraded to a →-operator only for configuration 3. The resulting process tree is therefore more of a general description of all behavior, than of a configurable process tree that also describes the behavior precisely.

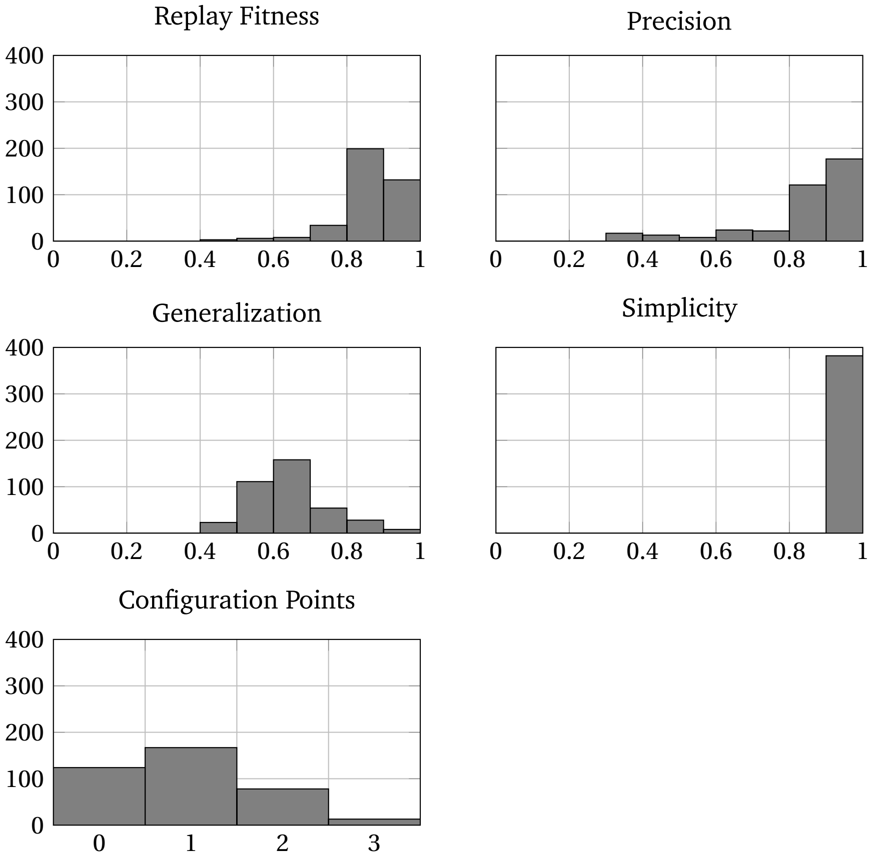


Figure 9.13: Distribution of the 382 candidates in the Pareto front over the quality dimensions for the running example.

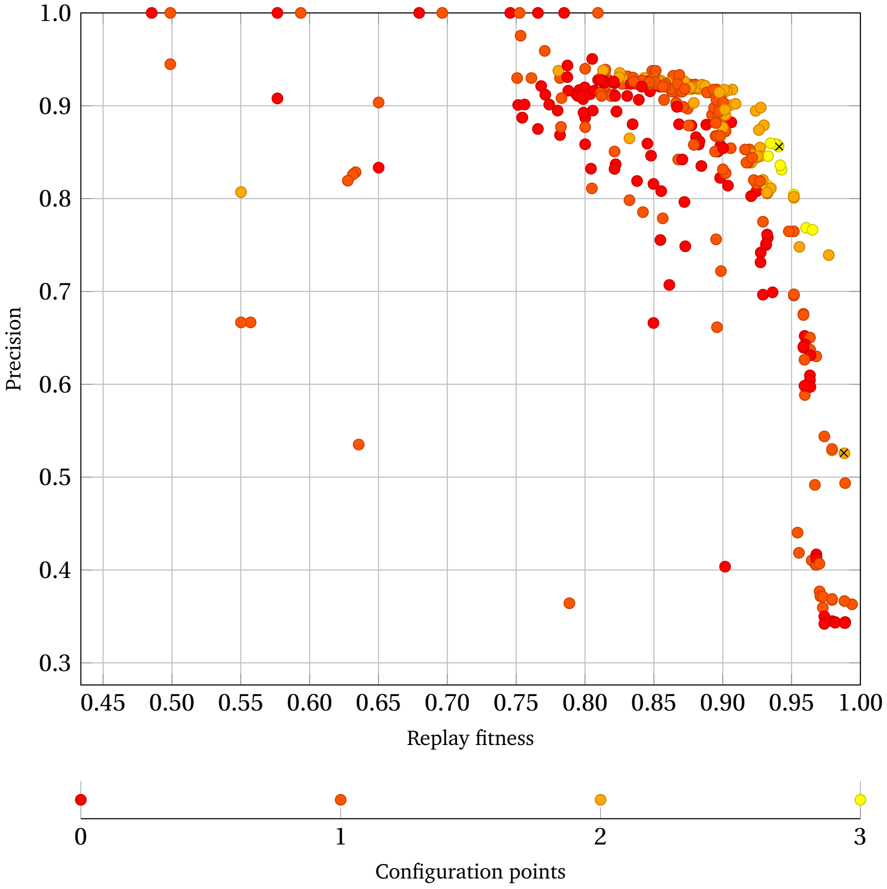
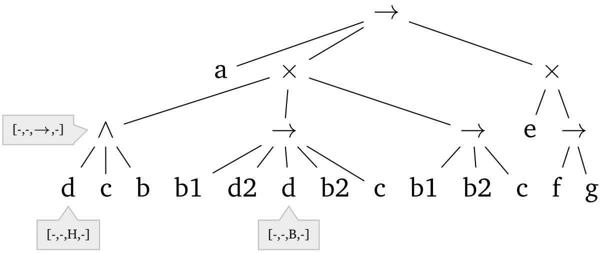


Figure 9.14: Visualization of the Pareto front of configurable process trees discovered by the ETMc algorithm on the running example event logs. The ×-symbols indicate the process trees shown in Figure 9.15 and Figure 9.16.

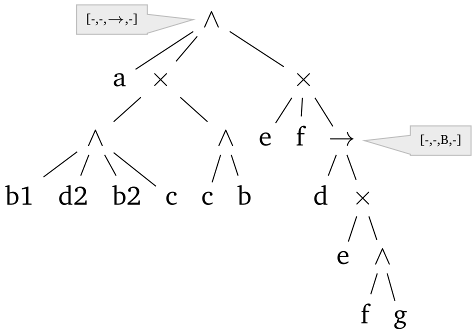


(a) Configurable process tree.

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.892	0.941	0.856	1.000	0.540	22	3	-
Variant 1	0.908	0.970	0.852	1.000	0.483	22	0	1.000
Variant 2	0.817	0.839	0.813	1.000	0.434	22	0	1.000
Variant 3	0.925	0.971	0.881	1.000	0.613	15	3	0.784
Variant 4	0.879	0.921	0.841	1.000	0.524	22	0	1.000

(b) Quality statistics of the configurable process model of (a).

Figure 9.15: Configurable process tree in the Pareto front discovered by Approach 4 for the running example event logs that balances replay fitness and precision.



(a) Configurable process tree.

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.836	0.988	0.526	1.000	0.572	21	2	-
Variant 1	0.822	0.969	0.541	1.000	0.592	21	0	1.000
Variant 2	0.746	1.000	0.244	1.000	0.455	21	0	1.000
Variant 3	0.903	1.000	0.750	1.000	0.593	14	2	0.771
Variant 4	0.748	0.977	0.272	1.000	0.589	21	0	1.000

(b) Quality statistics of the configurable process model of (a).

Figure 9.16: Configurable process tree in the Pareto front discovered by Approach 4 for the running example event logs that has good replay fitness at the cost of precision.

9.5.7 Comparison of the Four Approaches

The four approaches yield very different configurable process trees when applied to the running example event logs. The first two approaches are able to discover a configurable process tree that scores high on replay fitness and precision. However, the resulting configurable process tree is large and is mainly a global choice between the four input process trees.

The resulting configurable process trees of Approaches 3 and 4 are significantly smaller than the results of the first two approaches. The third approach, which first discovers a process tree and discovers the configurations in the subsequent phase, results in a configurable process tree with relatively high replay fitness, at the cost of precision and generalization. The fourth approach, the ETM*c* algorithm, balances all quality dimensions better, scoring high on replay fitness, precision and generalization. At the same time the number of configuration points is kept low, and the similarity of the process tree variants to the configurable process tree is high.

The Pareto front discovered by Approach 4, the ETM*c* algorithm, allows for further investigation of the trade-offs between the different quality dimensions. The process trees in the Pareto front have few configuration points, at most three, while there are four input event logs. Furthermore, configurable process trees that explain most of the behavior (as shown in Figure 9.16) are also discovered, ensuring high replay fitness.

The first two approaches seem to struggle with merging process models based on their behavior. Because they only focus on the structure of the model, the frequencies of parts of the process model being visited are not considered during the merge. The third and fourth approach both directly consider the behavior and frequencies as recorded in the event log. This seems to be beneficial for building a configurable process model since these latter two approaches outperform the first two in terms of size of the configurable process tree.

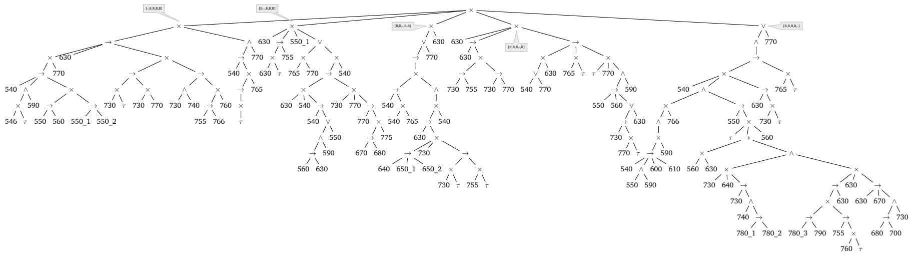
9.6 Case Study

To validate our findings we use a collection of five event logs from the CoSeLoG project, each describing a different process variant. The main statistics of the event logs are shown in Table 9.2. The event logs are extracted from the IT systems of five different municipalities. The process considered deals with objections related to building permits. Earlier, we already applied the ETM*d* algorithm on each of these event logs in isolation (see Section 7.4).

Table 9.2: Case study event log statistics

	#traces	#events	#activities
Combined	1,214	2,142	28
WABO1_BB	54	131	15
WABO2_BB	302	586	13
WABO3_BB	37	73	9
WABO4_BB	340	507	9
WABO5_BB	481	845	23

Both Approach 1 (merging individually discovered process models) and Approach 2 (merging similar discovered process models) result in large configurable process trees. These are shown in Figure 9.17 and Figure 9.18 respectively. Both models are large and cannot be understood easily, and again consist of an ×-operator as the root with each of the five original models as their chil-



(a) Configurable process model.

(b) Quality statistics of the configurable process model discovered using Approach 1 on the case study.

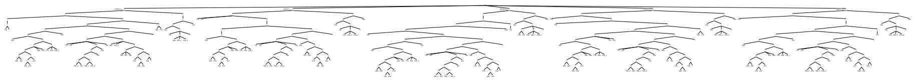
	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.953	0.989	0.973	0.946	0.460	215	5	-
Variant 1	0.941	0.989	0.976	0.913	0.328	46	4	0.352
Variant 2	0.946	0.978	0.974	0.947	0.481	38	4	0.300
Variant 3	0.938	1.000	0.954	0.867	0.313	30	4	0.245
Variant 4	0.961	1.000	0.980	0.944	0.490	36	4	0.287
Variant 5	0.949	0.988	0.967	0.957	0.452	69	4	0.486

Figure 9.17: Results of Approach 1, merging separate discovered process models, on the case study event logs.

dren that are then blocked, similar to the running example results. The statistics in Figure 9.17b and Figure 9.18b show that the replay fitness and precision scores are relatively high. However, the generalization and similarity scores of these process tree variants are very low. This is mainly caused by the unnecessarily large configurable process trees.

Approach 3, where the ETMc algorithm first discovers a common process model that is not very precise, and then applies configuration options, results in the process tree shown in Figure 9.19a. The resulting configurable process tree is significantly smaller than the results of the other two approaches. In total five configuration points are discovered to configure the process model for the individual event logs. Except for variant 2, for which a large part of the process tree is blocked, similarity scores are high.

Approach 4 results in the process tree shown in Figure 9.20a. In this approach the ETMc algorithm discovers the control flow and configuration points at the same time. The statistics are shown in Figure 9.20b. With only one configuration point, and better quality scores for all of the four quality dimensions than the result of Approach 3, this process tree is even smaller and hence simpler. The discovered configurable process tree scores significantly better on precision and generalization. This is mainly due to the more restrictive operators and less activity duplication. The discovered process tree does show that



(a) Configurable process model.

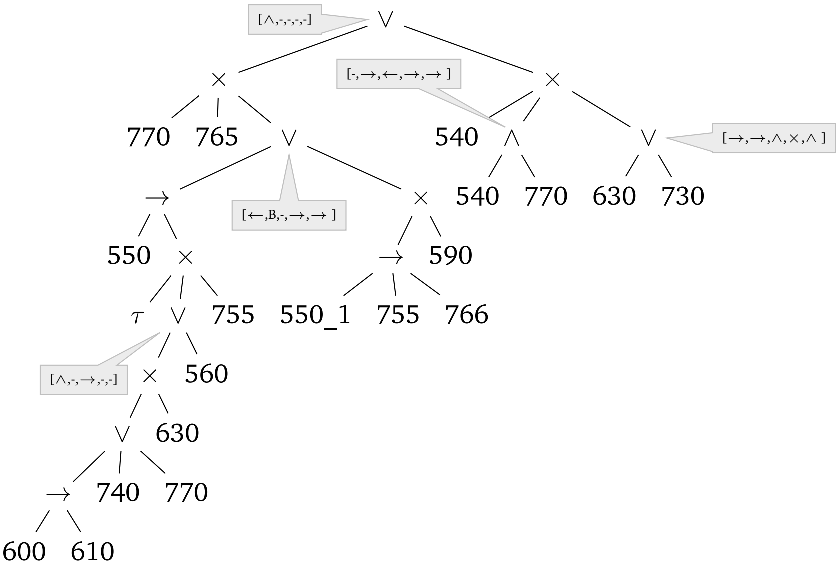
(b) Quality statistics of the configurable process model discovered using Approach 2 on the case study.

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.937	0.987	0.933	0.962	0.329	376	5	-
Variant 1	0.901	0.991	0.852	0.963	0.187	80	4	0.351
Variant 2	0.919	0.965	0.953	0.958	0.259	71	4	0.318
Variant 3	0.913	1.000	0.903	0.923	0.091	78	4	0.344
Variant 4	0.930	0.998	0.912	0.974	0.300	77	4	0.340
Variant 5	0.943	0.992	0.947	0.959	0.429	74	4	0.329

Figure 9.18: Results of Approach 2, merging the similar process models, on the case study event logs.

there are few differences between the five variants, since the process model can explain all five variants with high quality and only one configuration point.

The Pareto front that is constructed by the ETMc algorithm is limited to contain 200 configurable process trees. After 10,000 generations and normalization 44 process trees remain. Figure 9.21 shows the distribution of these 44 candidates over the different quality dimensions. Most configurable process trees



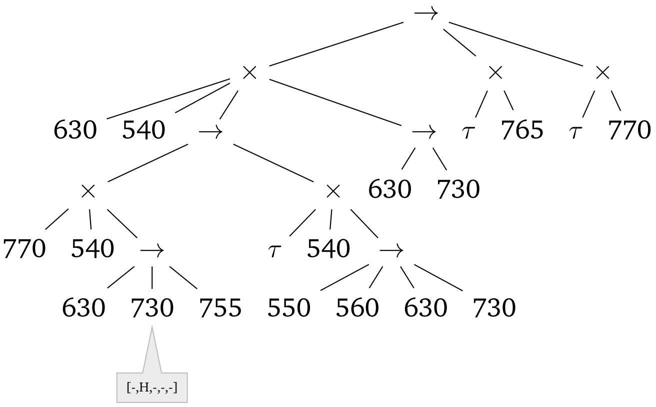
(a) Configurable process tree.

(b) Quality statistics of the configurable process model discovered using Approach 3 on the case study.

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.918	0.961	0.920	0.972	0.539	34	5	-
Variant 1	0.881	0.948	0.842	1.000	0.290	34	4	0.941
Variant 2	0.950	0.948	0.979	1.000	0.768	12	3	0.478
Variant 3	0.831	0.927	0.729	1.000	0.210	34	3	0.956
Variant 4	0.927	0.984	0.917	0.941	0.394	34	3	0.956
Variant 5	0.920	0.957	0.909	0.971	0.551	34	3	0.956

Figure 9.19: Results of Approach 3, the two-phase mining approach, on the case study event logs. The resulting configurable process tree is smaller than the results of Approaches 1 and 2.

have only one configuration point set. The Pareto front projected on the quality dimensions of replay fitness and precision is shown in Figure 9.22. This shows that most configurable process trees are precise, but that high replay fitness is hard to achieve. The configurable process tree with the best overall score for replay fitness is shown in Figure 9.23. The resulting configurable process tree has 95 nodes to precisely describe the observed behavior, as is shown by the many → and ×-operators. Only one configuration point is set, for variant 5, which removes an option in a choice. The resulting configurable process tree is not very precise, which might be solved by adding more configuration points. A smaller configurable process tree, that has a better trade-off between replay



(a) Configurable process tree.

(b) Quality statistics of the configurable process model discovered using Approach 4 on the case study.

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.952	0.966	0.968	1.000	0.672	29	1	-
Variant 1	0.893	0.913	0.921	1.000	0.448	29	0	1.000
Variant 2	0.959	0.980	0.962	1.000	0.704	28	1	0.982
Variant 3	0.905	0.948	0.894	1.000	0.435	29	0	1.000
Variant 4	0.960	0.986	0.978	1.000	0.573	29	0	1.000
Variant 5	0.949	0.949	0.975	1.000	0.765	29	0	1.000

Figure 9.20: Results of Approach 4, the integrated mining approach, on the case study event logs. The resulting configurable process tree is smaller than the result of Approach 3 while scoring better on replay fitness, precision, generalization and similarity.

fitness and precision, is shown in Figure 9.24. Although it has a lower score for replay fitness, it has a very high score for precision, while it is also a very small model. This configurable process tree also contains just one configuration point, this time for variant 3.

The application of the different approaches on the real-life event logs shows similar results as on the running example. The first two approaches seem to have difficulties in merging the process models based on the behavior of the process model. The third approach has problems in finding a precise configurable process tree. The fourth approach was able to find a good quality configurable process tree. The Pareto front discovered by Approach 4 showed various process trees with different trade-offs.

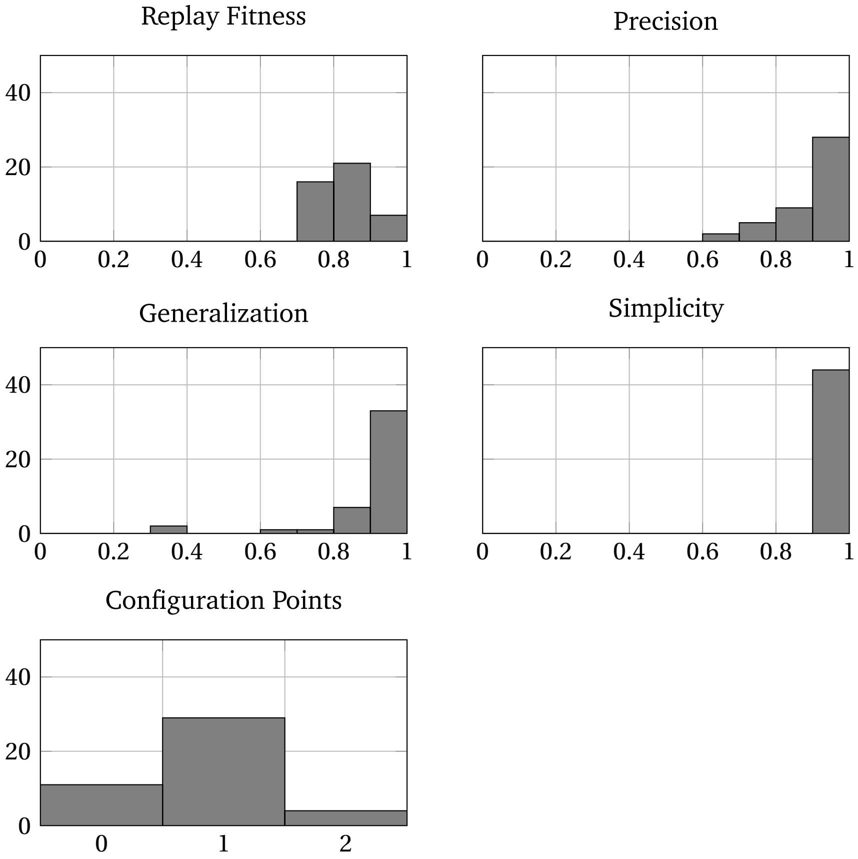


Figure 9.21: Distribution of the 44 candidates in the Pareto front for the case study over the quality dimensions.

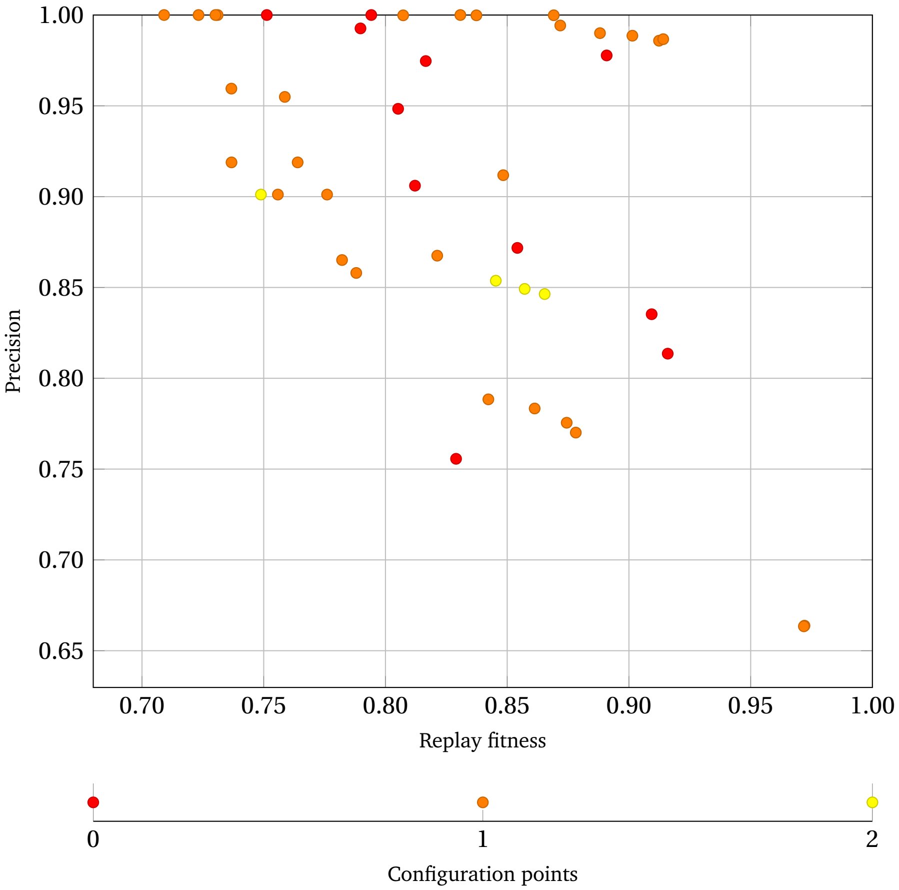
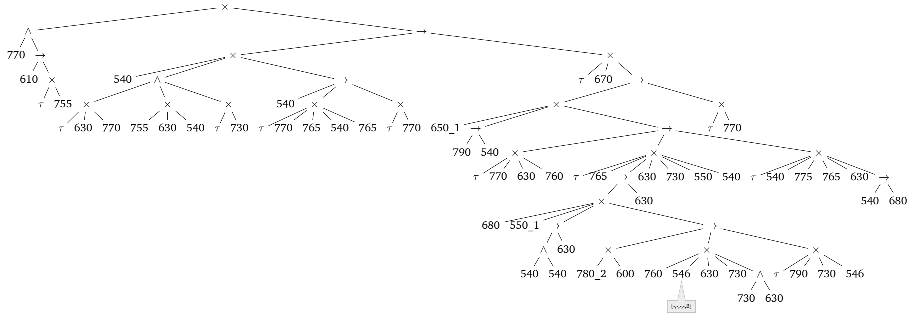


Figure 9.22: Visualization of the Pareto front of configurable process trees discovered by the ETMc algorithm on the case study event logs.

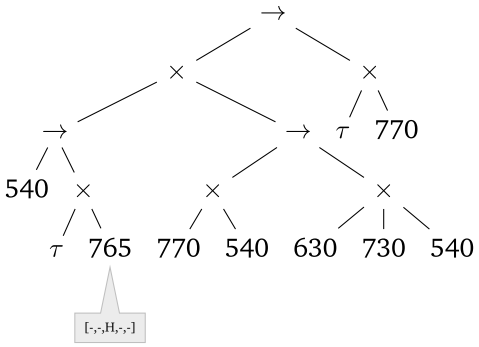

×
∧ →
770 → ×
610 × 540 ∧ → ×
τ 755 × × × 540 × × τ 670 →
τ 630 770 755 630 540 τ 730 τ 770 765 540 765 τ 770 × ×
650_1 → → τ 770
790 540 × × ×
τ 770 630 760 τ 765 → 630 730 550 540 τ 540 775 765 630 →
× 630 540 680
680 550_1 → →
∧ 630 × × ×
540 540 780_2 600 760 546 630 730 ∧ τ 790 730 546
730 630
[-,-,-,-,B]

(a) Configurable process tree.

(b) Quality statistics of the configurable process model of (a).

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.859	0.972	0.664	1.000	0.319	95	1	-
Variant 1	0.790	0.928	0.582	1.000	0.246	95	0	1.000
Variant 2	0.850	0.990	0.644	1.000	0.338	95	0	1.000
Variant 3	0.824	0.977	0.611	1.000	0.184	95	0	1.000
Variant 4	0.861	0.984	0.704	1.000	0.272	95	0	1.000
Variant 5	0.837	0.957	0.661	1.000	0.358	94	1	0.995

Figure 9.23: Configurable process tree found by Approach 4 for the case study event logs that has the best overall replay fitness (0.972).



(a) Configurable process tree.

(b) Quality statistics of the configurable process model of (a).

	Overall	Fitness	Precision	Simplicity	Generalization	Size	#C.P.	Similarity
Combined	0.937	0.914	0.987	0.997	0.836	18	1	-
Variant 1	0.872	0.831	0.959	1.000	0.714	18	0	1.000
Variant 2	0.934	0.909	0.992	1.000	0.830	18	0	1.000
Variant 3	0.852	0.843	0.914	0.889	0.600	18	1	0.972
Variant 4	0.960	0.951	0.993	1.000	0.837	18	0	1.000
Variant 5	0.933	0.906	0.987	1.000	0.870	18	0	1.000

Figure 9.24: Configurable process tree found by Approach 4 for the case study event logs with best trade-off between replay fitness and precision.

9.7 Related Work

Configurable process models can be constructed in different ways. They can be designed from scratch, but if a collection of existing process models already exists, a configurable process model can be derived by merging the different variants. The original models used as input correspond to configurations of the configurable process model.

Different approaches exist to merge a collection of existing process models into a configurable process model. A collection of EPCs can be merged using the technique presented in [91]. The resulting configurable EPC may allow for additional behavior, not possible in the original EPCs. La Rosa et al. [118] describe an alternative approach that allows merging process models into a configurable process model, even if the input process models are in different formalisms. In such merging approaches, some configurations may correspond to an unsound process model. Li et al. [124–126] discuss an approach where an existing reference process model is improved by analyzing the different variants derived from it. However, the result is not a configurable process model but an improved reference process model, i.e., variants are obtained by modifying the reference model rather than by process configuration. The CoSeNet approach [161] has been designed for merging a collection of block-structured process models. This approach always results in sound and reversible configurable process models. Since the CoSeNet approach works on a structure similar to process trees, this approach is used as the process model merging algorithm in the experiments discussed in this chapter.

Another way of obtaining a configurable process model is not by merging process models but by applying process mining techniques on a collection of event logs. This idea was first proposed in [92] where two different approaches were discussed, but these were not supported by concrete discovery algorithms. In this chapter we implemented both approaches as Approach 1 and Approach 3. Assy et al. [30] propose to mine configurable business process fragments. Their approach is twofold: first sublogs are extracted around selected activities from which configurable process fragments are discovered in a second phase. All extracted sublogs for an activity are merged and fragments are discovered on these merged sublogs. The discovered fragments are then merged into a single configurable process model. Additionally, configuration guidelines are deduced. Although extracting several sublogs reduces the complexity of the individual problems to be solved, it also introduces new problems. For instance, behavior is not replayed but (un)shared activities are detected. This approach does not work when the extracted sublogs around the selected activities are too small or too varied, which is an issue in the case of choices, parallelism or loop constructs consisting of many activities. Furthermore, it is unclear how fragments should be merged when there is no overlap or when multiple merging options exist.

9.8 Conclusion

In this chapter we presented four approaches to addressing Challenge 6 which states that a family of processes should also be described. In this chapter we first discussed several configurable process model notations that are able to represent a family of processes. We then extended the process tree notation to support configurations which reduce the allowed behavior of a process tree. Four approaches were presented in this chapter in order to discover a configurable process model from a collection of event logs. The fourth approach applies the ETM*c* algorithm which adds configuration mutation and change operations to the ETM*d* algorithm. All four approaches have been applied, and their results have been compared, on both a running example and a real life data set. The results showed that merging process models produces larger, and therefore more complex, process models. Discovering a configurable process model from the event logs results in smaller process models with fewer configuration points. At the same time, each of the individualized process trees score high on all four quality dimensions.

Chapter 10

Inter-Organizational Process Comparison

In this chapter we address Challenge 7: “Compare similar observed behavior”. In Chapter 9 we discussed several ways in which a configurable process model can be discovered from a collection of event logs. The discovered configurable process model describes a family of processes, based on the individual event logs. However, no insights into differences such as throughput time or a more detailed comparison of the observed behavior are provided.

In this chapter we present a new analytical technique that allows for a *dual* comparison. In the first place, it allows for a comparison between the intended and the actual execution of a business process. Secondly, it supports the comparison of various parties (organizations) executing that same process. These

This chapter is based on the work presented in [47, 54]

- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Towards cross-organizational process mining in collections of process models and their executions. In F. Daniel, K. Barkaoui, S. Dustdar, W.M.P. van der Aalst, J. Mylopoulos, M. Rosemann, M. Shaw, and C. Szyperski, editors, *Business Process Management Workshops (2)*, volume 100 of *Lecture Notes in Business Information Processing*, pages 2–13. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-28115-0
- J.C.A.M. Buijs and H.A. Reijers. Comparing business process variants using models and event logs. In I. Bider, K. Gaaloul, J. Krogstie, S. Nurcan, H.A. Proper, R. Schmidt, and P. Soffer, editors, *BMMDS/EMMSAD*, volume 175 of *Lecture Notes in Business Information Processing*, pages 154–168. Springer, 2014. ISBN 978-3-662-43744-5

comparisons are visualized through a so-called *alignment matrix*. We also describe a *comparison framework* that shows the methodical application of this aid.

We introduce a *comparison table* between modeled and observed behavior and we extend it by explicitly incorporating the process model, i.e., the intended behavior, into the comparison. By replaying the actual behavior on that initial model, as witnessed through event logs, and showing where different organizations deviate, the process model can be used as a common means to compare against. This cross-comparison can help organizations get a better understanding of how a process is executed and act on that insight. Such actions may be diverse: It may be decided to fix the common process if it allows for too much deviation, but individual organizations may also want to imitate the practices of another partner when these seem preferable.

The presented comparison framework is evaluated using a case study with five of the municipalities participating in the CoSeLoG project (see Section 1.4). These municipalities have decided to start working more closely together while maintaining their legal autonomy. After having executed a commonly designed process for a prolonged amount of time, they have an interest in the type of comparison we sketched earlier: How is each organization carrying out this process and how do they differ from each other in different respects?

The remainder of this chapter is organized as follows. First, a running example is introduced in Section 10.1. Section 10.2 presents the comparison framework used to compare behaviors of different processes and organizations. In Section 10.3 the alignment matrix is presented that visualizes alignments of observed behavior on process models. The overall comparison approach is applied in a case study, which is described in Section 10.4. In Section 10.5 we reflect on related work that also aims to analyze and compare processes. Section 10.6 summarizes our findings and conclusions.

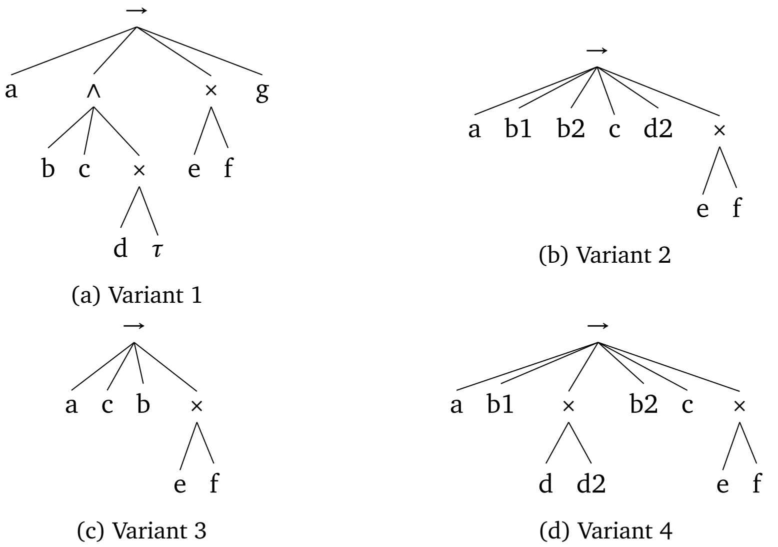
10.1 Running Example

We use the running example of Section 1.2.2 to illustrate our approach. The running example consists of four process model variants, shown in Figure 10.1, and four corresponding event logs, as shown in Table 10.1. All four variants describe the process for handling loan applications. Even though the processes differ slightly, each process sends an e-mail (activity `a`) and in the end either accepts (activity `e`) or rejects (activity `f`) the application, followed by sending the decision via e-mail (activity `g`) for variant 1. The order in which the other activities can be executed differs as well. Moreover, each variant differs as to which activities are included. For instance, either activity `b` is part of the variant, or both activities `b1` and `b2`, which are more fine-grained, are included. The corresponding event logs describe possible executions of the corresponding process model. Please note that in this example the traces align perfectly with the corresponding process model. This is generally not the case for real-life processes. However, we can always use alignments to squeeze observed behavior into the process models.

10.2 Cross-Organizational Comparison Framework

In order to compare processes between organizations we propose a *cross-organizational comparison framework*. The framework aims to facilitate a comparison of business processes by using both the process models *and* the observed behavior.

The general approach of the comparison framework is shown in Table 10.2.



(a) Variant 1

(b) Variant 2

(c) Variant 3

(d) Variant 4

Figure 10.1: The process trees of the four variants of the running example (see Section 1.2.2).

Table 10.1: Four event logs for the four different variants of the loan application process of Figure 10.1.

(a) Event log for variant 1

Trace	#
a b c d f g	38
a b d c f g	26
a b d c e g	12
a b c f g	8
a b c d e g	6
a d c b f g	4
a c d b f g	2
a c b e g	1
a d b c f g	1
a d b c e g	1
a c b f g	1

(b) Event log for variant 2

Trace	#
a b1 b2 c d2 f	50
a b1 b2 c d2 e	20

(c) Event log for variant 3

Trace	#
a c b e	120
a c b f	80

(d) Event log for variant 4

Trace	#
a b1 d2 b2 c f	60
a b1 d b2 c e	45

Table 10.2: The comparison table of the comparison framework approach. Event logs L_1 through L_n and process models M_1 through M_m are used as input. Three types of metric are shown: (event)log metrics, model metrics and comparison metrics.

	M_1	...	**M_m**	**Log Stat**
L_1	compareMetric(L_1,M_1)	...	compareMetric(L_1,M_m)	logMetric(L_1)
⋮	⋮	⋱	⋮	⋮
L_n	compareMetric(L_n,M_1)	...	compareMetric(L_n,M_m)	logMetric(L_n)
Model Stat	modelMetric(M_1)	...	modelMetric(M_m)	

The main result of the comparison framework is a *comparison table* which compares event logs and process models. The input of the comparison framework is one or more event logs, which are displayed as rows in the comparison table of Table 10.2. Additionally, one or more process models are used and displayed as columns in the comparison table. An event log metric ('logMetric') can be selected and the result is visualized for each individual event log in the comparison table. Similarly, a process model metric ('modelMetric') is displayed for each individual process model in the comparison table. For each combination of event log and process model a comparison metric ('compareMetric') is displayed. Note that the term metric is a bit stretched in this chapter in the sense that a metric might produce a graphic such as a chart or a graph.

Within the framework several metrics exist for each of the three types. The user can select which one to use for each type.

10.2.1 Process Model Metrics

Process model metrics are metrics calculated using only the process model. Many metrics based on process models exist [130], for example metrics that measure size, density, partitionability, connector interplay, cyclicity or concurrency. Another example is metrics that measure simplicity, as discussed in Section 5.3.

Currently the comparison framework provides two process model metrics: process model size and a visualization of the process model. The main focus of the comparison framework is the inclusion of the observed behavior in the comparison. However, additional process model metrics can easily be added, as is explained in Section 11.3.3.

10.2.2 Event Log Metrics

Event log metrics can range from simple metrics counting the number of traces and events, to more general metrics related to different performance indicators. Currently over ten event log metrics are included in the comparison framework, which include number of traces, average trace duration, occurrence frequency per activity and social network.

10.2.3 Comparison Metrics

The third category of metrics is *comparison metrics*. This relates to comparisons between an event log and a process model. The alignments discussed in

Table 10.3: Application of the comparison framework on the running example, with the event log metric set to number of traces, the process model metric to the number of nodes in the model and the comparison metric to the replay fitness.

	Config 1	Config 2	Config 3	Config 4	Log Stat
Event Log 1	1.000	0.506	0.575	0.580	100
Event Log 2	0.525	1.000	0.553	0.833	70
Event Log 3	0.933	0.656	1.000	0.656	200
Event Log 4	0.579	0.833	0.553	1.000	105
Model Stat	12	9	7	11	

Section 5.4.1 are an example of a comparison metric, as are the precision and generalization metrics discussed in Chapter 5. The alignment matrix that we present in Section 10.3 is currently the most advanced comparison metric.

10.2.4 Application on the Running Example

An application of the comparison framework on the running example is shown in Table 10.3. The specific process model metric chosen here is the number of nodes in the process model. The number of cases in the event log defines the event log metric. Each comparison cell displays the replay fitness score (see Section 5.4.1), calculated on the alignments. Higher values indicate better alignments, which are emphasized by increasingly darker shades of green as background color.

When comparing the size of the event logs, one can see that event log 3 has the most traces, and event log 2 contains the fewest traces. The process model metric indicates that organization 3 with 7 nodes has the smallest process model, while organizations 1 and 4 have the biggest process models, with 12 and 11 nodes respectively. When we investigate the replay fitness scores, we can see that the diagonal has a perfect score of 1.000. This means that the process model of each organization perfectly explains the observed behavior. Furthermore, the process model of organization 1 describes the observed behavior of organization 3 quite well. However, the process model of organization 3 does not explain the observed behavior of any of the other organizations very well. Organizations 2 and 4 have reasonable replay fitness scores on each other's process models, which might allow these organizations to start a collaboration.

Trace	a	b	c	≫	d	e	g
Model	a	≫	c	b	≫	e	≫

(a) Alignment between the trace $\langle a, b, c, d, e, g\rangle$ from event log variant 1 and the process model of variant 3.

	a	b	c	d	. . .
Alignment 1	(a,a)	(b,≫) (≫,b)	(c,c)	(d,≫)	. . .
⋮	⋮	⋮	⋮	⋮	⋱
Alignment 100	(a,a)	(b,b)	(c,c)		. . .

(b) A concrete instance of the alignment matrix with alignment steps shown in the cells.

Figure 10.2: Alignments and the construction of the alignment matrix.

The simple replay fitness scores give some preliminary insights, but do not provide a deep understanding of the level of similarity of behavior between the different organizations. To provide more in-depth insights we propose the alignment matrix visualization as a comparison metric.

10.3 Visualizing Alignments: the Alignment Matrix

The purpose of the *alignment matrix visualization* is to visualize the alignments, calculated using both the process model and the event log, in a concise but clear way. However, we do not project alignments on either the event log or the process model. Instead, we want to exploit the utilization of the available space, whether this concerns a digital display or a physical canvas, to allow for a wider exploration. Furthermore, we synchronize the settings of the different alignment matrices to ensure all matrices are indeed comparable.

The input for the alignment matrix consists of the alignments for the traces of the event log. Figure 10.2a shows such an alignment between a trace from the event log and a completed trace of the process model. The alignment consists of several *alignment steps*. Each alignment step contains information about which trace and process model it relates to. It also contains a relation to an event in that trace, to an activity in the process model, or both.

Within the alignment matrix, alignment steps are assigned to one or more cells, which are distributed over columns and rows. An example is shown in

Figure 10.2b. Here, the columns are defined to be the activities, while each row is an alignment instance. In this way, each cell contains those alignment steps that for a particular alignment are related to a certain activity.

Other settings for the column and row definitions are possible, for instance, changing the rows to represent the different users in the process, and the columns to represent a day or week each. This visualizes when certain users are active and if they execute the activities according to the process model.

In the example of Figure 10.2b, most cells contain one alignment step. An exception is the cell for trace 1 and activity b. Since the alignment contained both a log move and a model move on this activity, this cell contains two alignment steps. Furthermore, since trace 100 (the last trace of event log variant 1 of Table 10.1) did not contain activity d and the process model did not enforce the execution of this activity, the corresponding cell is empty.

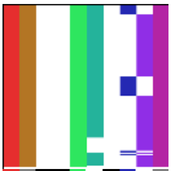
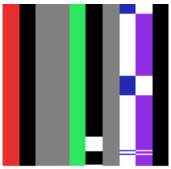
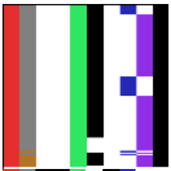
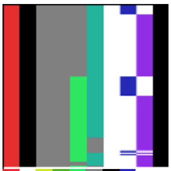
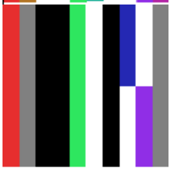
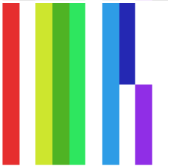
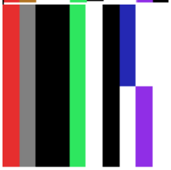
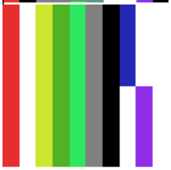
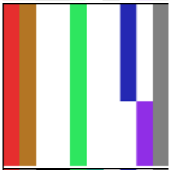
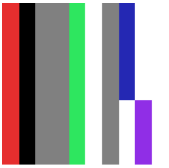
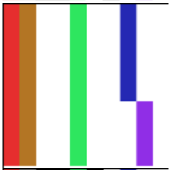
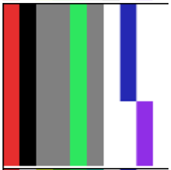
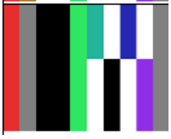
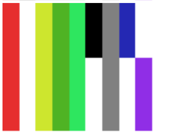
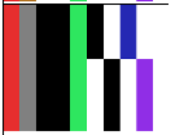
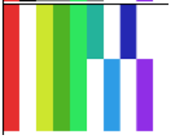
Since in general there can be many alignment steps in a cell, we do not show these individual steps. Instead, we aggregate them and express them with various colors:

- If the cell is empty, i.e. there are *no alignment steps*, we color the cell white;
- In case the cell mainly contains *log move steps*, we color the cell black;
- If the cell mainly contains *model move steps* we color that cell gray;
- In case the cell mainly contains *synchronous steps*, we color the cell according to a pre-defined color that is assigned to that activity (red, yellow, green, blue, purple, etc.).

An application of the comparison framework using exactly the settings as discussed is shown in Table 10.4. Here, the four event logs of Table 10.1 are replayed on the four process models of Figure 10.1. Each of these replays is visualized using an alignment matrix. The columns in each alignment matrix represent the activities (a through g), while each row is a single alignment of a trace.

Let us examine, for the example, the replay of event log 1 on process model variant 3. It shows both black and gray cells, which indicate mismatches, log move and model move steps respectively. It can be seen that activity a can be replayed correctly, as indicated by the red color. The gray column, however, indicates that activity b cannot be replayed correctly, except in the last couple of traces as visualized by the orange color in that column.

Table 10.4: Application of the alignment matrices on the running example. Each column of the comparison framework represents a process model variant from Figure 10.1 and each row an event log from Table 10.1. Inside each cell an alignment matrix is shown where the columns are activities, the rows are traces and the color is determined by move type and activity. Event log metrics and process model metrics are not shown here.

	Variant 1	Variant 2	Variant 3	Variant 4
Event log 1				
Event log 2				
Event log 3				
Event log 4				

We can now also further investigate the previous observation that the process model of organization 1 seems to match quite well with the observed behavior of organization 3. The alignment matrix of this combination shows mainly white and colored columns, but the last column for activity g is completely gray. This indicates that activity g is always a move on model only. Therefore, if the process model of organization 1 simply allows for the option to skip activity g, the same process model can be used without any problems by organization 3. In other words, these organizations basically work in the same way, which could be exploited in various ways.

10.4 Case Study

In order to validate our comparison framework we applied it to a building permits process. Five municipalities from the CoSeLoG project are collaborating on the building permits process and jointly selected and configured an information system to support this process. However, the five municipalities use their own instances of the system with slightly different settings for each. Moreover, the system allows for some flexibility during the execution of the process. Because of these reasons, several differences still exist in the way the municipalities execute the process. The long-term goal of the municipalities is to centralize and standardize the process to reduce costs, but this goal can only be attained by making gradual steps. For this reason, it is crucial for the municipalities to understand individual differences between these processes and address them one by one.

In this section we describe the set-up of the case study and how it was executed. We also provide the insights that we extracted from it.

10.4.1 Setup

We planned a meeting in February 2014 and invited representatives of each of the five involved municipalities. The meeting was set up to consist of two parts. The aim of the first part was to present general information (number of cases and average throughput time), together with dotted chart [165] and social network [166] visualizations of cases from 2013, detailed along different case types. No process models or activity details were given in that part. Roughly one hour was devoted to this first part.

In the second part, planned to cover approximately another hour, we set out to explain the global idea of the comparison table, i.e., comparing the behavior of a municipality with the discovered model from the behavior of another municipality. The idea for this part was to show the comparison table with the replay fitness scores, as shown in Figure 10.3a. This was then followed by an example of the alignment matrix (the matrix of event log 1 on variant 4 from Table 10.4). During a small break of 5 minutes we laid out the 25 (5 by 5) printouts of the alignment matrices on a table. After the break the idea was to gather everyone around the table and provide each participant with an individual color marker. In this way, each participant could mark observations on the printouts. During this part, participants were stimulated to make observations and initiate discussion.

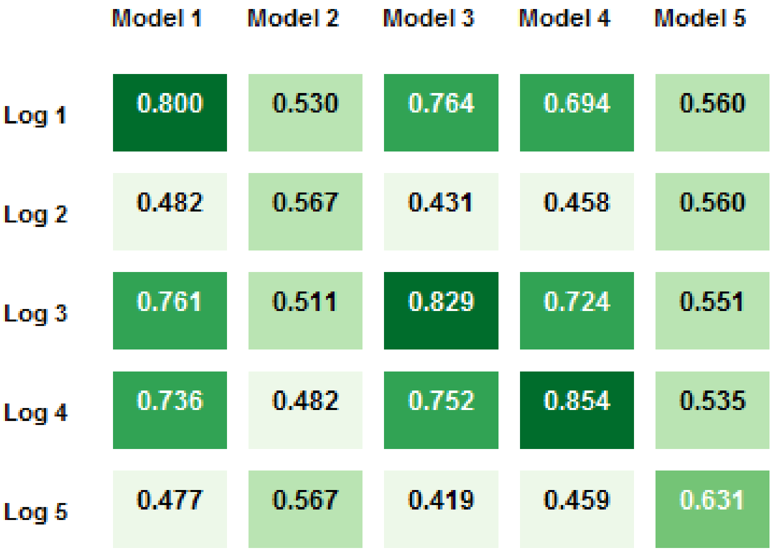
We invited seven representatives for the meeting, of which six eventually joined. The expertise from all participating municipalities was present except for municipality 2, whose representative was unable to attend. Fortunately, the representative of municipality 5 also had knowledge of the process in municipality 2. Two representatives were present for municipalities 1 and 3. For each of these two municipalities a coordinator of the process within their respective municipality was present, and both also collaborated in the process. For municipality 4 a building permits expert working in the process was present. As such, these three people had a very good understanding of the whole process. The remaining three representatives were a coordinator of automation and internal affairs (municipality 1), a specialist on internal control and electronic services (municipality 3), and a policy officer for environmental law (municipality 5). As such, these three people had a more high-level understanding of the whole process, but also detailed knowledge of (parts of) the process.

The event logs used in the case study contain cases that were started at some point in 2013 within any of the five municipalities. The logs cover between 150 and 300 cases for each of the municipalities. Both the event logs and the process models contain the 47 most frequent activities across all municipalities. The process models used were automatically discovered using the ETM*d* algorithm based on the data of the event logs. The reason for this is that the municipalities in question immediately configured the information system to their individual preferences without the use of an explicit process model. While the logic of a configuration setting in principle could be translated in a process model, we opted for the use of the discovered model as a reasonable proxy for it.

10.4.2 Execution

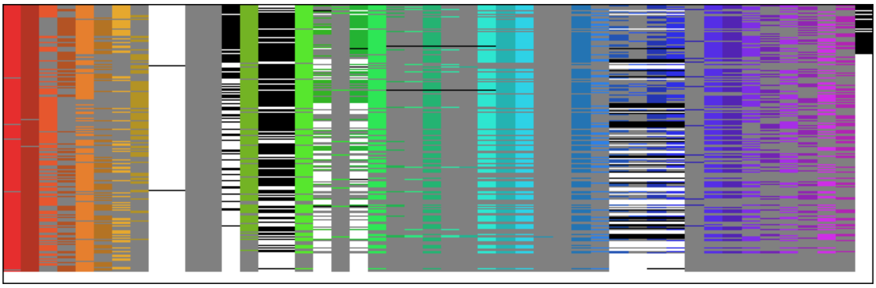
First, before showing the alignment matrices to the representatives, we showed the replay fitness table as shown in Figure 10.3a. We first explained that each number roughly corresponded to the fraction of correctly explained events. The participants quickly noticed that these ratios were not overly high, and in many combinations even very low. They also noticed differences between municipalities. After being asked if they could identify distinct clusters of municipalities they replied they could recognize a group consisting of municipalities 1, 3 and 4 which is likely to display highly similar behavior.

Next a small break was introduced and the 25 alignment matrices were distributed on the table, where municipality 2 was moved between municipalities 4 and 5, so that 1, 3 and 4 (as a group of similar municipalities) were close



	Model 1	Model 2	Model 3	Model 4	Model 5
Log 1	0.800	0.530	0.764	0.694	0.560
Log 2	0.482	0.567	0.431	0.458	0.560
Log 3	0.761	0.511	0.829	0.724	0.551
Log 4	0.736	0.482	0.752	0.854	0.535
Log 5	0.477	0.567	0.419	0.459	0.631

(a) Comparison table shown with replay fitness scores (green) as comparison metric.



(b) Example of one of the 25 alignment matrices shown, more specifically of the behavior of municipality 5 on the model of municipality 1.

Figure 10.3: Two of the analysis results shown to the case study participants.

together. An example of one of the 25 alignment matrices is shown in Figure 10.3b.

After the break we gathered everyone around the laid-out alignment matrices, as is shown in Figure 10.4a. One of the first things that was noticed on the basis of the alignment matrices was that there was a considerable number of black cells, which the participants understood to be ‘bad’. One of the representatives of municipality 3 contributed that he noticed that each municipality had significantly fewer black cells on their own matrix, which can be expected.

Another observation made was that one municipality had a lot of white cells in the alignment matrix. From this, the participants concluded that the specific case types dealt with by this municipality could be different than those of the others, since they show different activities in their behavior.

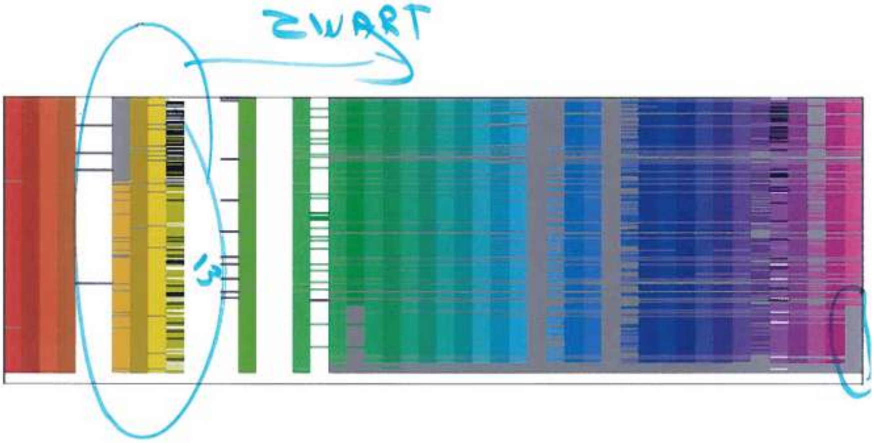
Based on the alignment matrix of municipality 4 on its own model three observations were made, as is shown in Figure 10.4b. The first observation, denoted by the bigger blue circle on the left, is that first of all there is not much black (‘zwart’ in Dutch) visible in this matrix. A second observation, made by two of the participants, was that in two of the columns there was a mix of color and black. They correctly concluded that this was caused by correctly executing this activity some of the time, and deviating from the process model at other times.

The third observation made on this matrix, as indicated by the blue circle in the bottom-right, is that the last activity shows a lot of gray for the last few traces. A brief remark by us that the cases were sorted from old in the top rows to the new in the bottom rows, quickly resulted in the correct conclusion that these cases did not reach that particular activity in the process just yet. Quickly after this, the participants observed that some of the newer cases actually were further along in the process. They expected a diagonal line from bottom left to upper right. They also noticed that this was not the case for all municipalities.

All-in-all, 22 observations were counted. Each of these triggered a discussion and an exploration of explanations for it between the participants. In the end, one participant remarked that he would like to rearrange the alignment matrices and only show the matrices of the replay on the municipality’s own process models. In this set-up, another 11 observations were made.

We then gently ended the discussion and asked the participants if they thought this approach was easy to understand and use. Although we noticed that the participants seemed somewhat overwhelmed during the introduction of the rather colorful pictures in the beginning, they did not mention this during the evaluation. All people involved noted that detecting the gray and black, and also white, worked well. Furthermore, the colors helped participants with

(a) Photo of the set-up and the participants. Participants faces are obscured except for the third person from the left which is the author of this thesis.



(b) The alignment matrix of municipality 4 on its own model, shown with annotations.

Figure 10.4: A photo and an annotated alignment matrix with some observations made during the case study.

relating parts of the process across alignment matrices. A further remark that was made was that the colors made it easy to distinguish between irregular behavior and more structured executions of the process, which was considered highly useful.

From a content perspective, the participants expressed satisfaction with what they could observe using the alignment matrices. From the various insights that were obtained by observing the alignment matrices of each other's processes, we provide two striking examples. The first of these relates to the observation that one municipality actually did not execute certain steps, while their products still adhered to the regulations. All people involved mentioned that it would be valuable to investigate whether this way of working could be adopted by all municipalities. Secondly, the participants could recognize the effect of a change of personnel within a particular municipality. They expressed that it would be interesting to keep following the execution of the process to see if the behaviorial differences caused by this would stabilize over time.

Improvement suggestions with respect to the comparison approach were also made. One suggestion was to add more *visual anchors*, which could help to better determine the location in the alignment matrices, i.e., to remember which column represented which part of the process, and for the rows in which month the case arrived. Also, the participants would prefer additional features for increased interactivity with the data. They were particularly interested in selecting specific case types to see how these compared across municipalities. Another interactive feature they proposed was selecting only certain employees, since they had the intuition that certain employees performed well (or badly), in particular municipalities.

10.4.3 Results

The goal of our visualization is to provide insights into the commonalities and differences in behavior between organizations. After a brief explanation of the alignment matrix, we let all participants observe and discuss based on alignment matrix printouts. We noticed that some picked up how to read the figures quicker than others, but after a few minutes almost all participants joined in the discussion. All but one of them regularly made observations, supported observations from others or came up with possible explanations for observations. Moreover, actual input from the organizers to clarify certain things was required only very infrequently after the initial explanation of the alignment matrix. Overall, we counted over 30 observations in about half an hour of discussion, which underscores how helpful the approach is in comparing the involved processes.

One of the main comments we noted was that the participants would favor more interaction opportunities with the visualization. By hovering over a cell they would like to see more details of that cell, such as the activity, resource and case involved. They also showed real interest in the ability to filter on case types and resources, in order to validate certain assumptions they may have.

However, the main thing we noticed was that the alignment matrices, and the comparison of process executions in general, triggered a lot of discussions between the participants. Participants often asked each other questions of the type “But how do you do this?”, or “Why are you faster?”, or “Does this role perform this type of activity?”. We see these as proof that the comparison approach triggers a meaningful discussion based on actual analysis results.

10.5 Related Work

For a considerable time now, organizations seek to learn from others how to adapt their own processes to improve competitiveness [153]. Process benchmarking, however, is primarily a manual process, requiring the involvement of experts to collect and interpret process-related data [168]. The main problem that has been recognized is that processes across different organizations are often modeled with different levels of granularity and for different purposes. This makes their comparison hard. Previous research in the area of process benchmarking [82,106,168] has mainly focused on semantic approaches to overcome these types of barriers.

In the context of our work, the processes that are to be compared can be considered variants of each other [127]. This means that the processes are different, but share essential characteristics through their conformance to a shared set of constraints [128] or their derivation from a common template [99]. Because of this starting point, their semantical matching is not really an issue. Yet, the emphasis of existing work on model variants is on the management, specification, and comparison of models, i.e., the design-time perspective of these processes. Our approach widens this scope by incorporating the actual behavior of these variants, i.e., the run-time perspective. In other words, we extend process model variant management with analytical approaches that allow for comparing the supposed/intended behavior of processes with their actual execution.

Two categories of approaches with respect to comparing the supposed or intended behavior with the actual behavior of a process can be identified. The first of these encompasses approaches that pursue delta analysis between a predefined process model on the one hand and the *discovered* model derived from event logs on the other [78, 109]. Here generic approaches also play a role that relates to process model matching, see [178]. The second category aims to project the actual behavior of a process onto the predefined process model, as in [111]. The aim is then to show how individual instances relate to pre-defined process model parts. Our research is most related to the latter category. In contrast to existing work, however, it specifically builds on the notion of process alignments (see Section 5.4.1). Another innovative angle in this context is our interest in the comparison of multiple, related processes.

Since our work strongly emphasizes the visualization of the analysis results, it also relates to other approaches that help make better sense of process models. These approaches cover the usability aspects of the employed notation [136], ways to emphasize the logical relations between model elements [151], and bringing in new perspectives [38], to name a few.

In summary, our work is at the intersection of the streams of analytical and visualization research to support process benchmarking across process variants. We extend existing work by taking both the supposed behavior and the actual behavior of the process variants into account.

10.6 Conclusion

In this chapter we presented the comparison framework that allows for a comparison of event logs and process models. The comparison framework shows metrics that give insights into the individual event logs, process models, and the combinations of these. This provides insights into (dis)similarities between organizations beyond the control flow perspective. Additionally we presented the alignment matrix comparison metric that aims to provide a quick comparison between the different event logs and process models. By synchronizing the settings between the alignment matrices, easy comparison is aided. Furthermore, both the comparison framework and the alignment metric are flexible and can be extended easily to incorporate new metrics. The applicability of the alignment matrix, as part of the encompassing comparison framework, was demonstrated using a case study.

Chapter 11

Implementation

The ETM framework and the ETM*d*, ETM*r* and ETM*c* algorithms are all implemented in the open-source framework for process mining ProM [174, 176]. All algorithms are implemented in the Evolutionary Tree Miner package, which is distributed with ProM version 6.4 and later[1,2]. In this chapter we discuss the implementation aspect of the ETM framework in more detail. Section 11.1 shows how an experiment from this thesis can be performed via the graphical user interface of ProM, via code and via command line calls. The ETM framework can easily be extended by adding new quality dimensions, quality metrics, change operations, and much more, as is discussed in Section 11.2. In Section 11.3 we discuss the implementation of the comparison framework presented in Chapter 10. Section 11.4 concludes this chapter.

11.1 Walk through of the ETM*d* Algorithm

In this section we demonstrate different uses of the ETM framework, and the ETM*d* algorithm in particular. We do this by replicating the experiment performed in Section 7.1.3 where a Pareto front is discovered on the running example with exceptional behavior. This experiment is first replicated via the

[1] A preliminary implementation is included in ProM 6.3 as a 'runner up'-package but significant improvements have been made since.

[2] Please note that this chapter was written before the official release of ProM 6.4, therefore the plug-ins and code described in this chapter can differ slightly from the released version of the ETM package. Up-to-date and extensive documentation will be provided upon release.

ProM graphical user interface (GUI) as is explained in detail in Section 11.1.1. In Section 11.1.2 we replicate this experiment via Java code calls. We then show how the experiment can be performed via command line interface (CLI) calls, which is discussed in more detail in Section 11.1.3.

11.1.1 Usage via the GUI

With ProM started and the event log of the running example with exceptional behavior loaded, we can search for all plug-ins containing 'ETM'. The result of this search is shown in Figure 11.1. A total of nine plug-ins have been found. Six of these can be run on an event log without additional input. The three plug-ins marked with yellow require additional input. Currently the 'Mine Pareto front with ETMd in Live mode' is selected, which produces a 'Running ETM*d* Live Pareto Instance', as is shown on the right-hand side of Figure 11.1.

The following nine ETM plug-ins are available:

1. **Mine a Process Tree with ETM*d***
   (input: event log; output: process tree)

2. **Mine a Process Tree with ETM*r***
   (input: event log, process tree(s); output: (repaired) process tree)

3. **Mine Configured Process Tree with ETMc**
   (input: event logs; output: configured process tree)

4. **Mine Pareto front with ETMc**
   (input: event logs; output: Pareto front of configured process trees)

5. **Mine Pareto front with ETMc in Live mode**
   *(input: event logs; output: live view of the ETM*c *algorithm constructing a Pareto front of configured process trees)*

6. **Mine Pareto front with ETM*d***
   (input: event log; output: Pareto front of process trees)

7. **Mine Pareto front with ETM*d* in Live mode**
   *(input: event log; output: live view of the ETM*d *algorithm constructing a Pareto front or process trees)*

8. **Mine Pareto front with ETM*r***
   (input: event log, process tree(s); output: Pareto front of (repaired) process trees)

9. **Mine Pareto front with ETM*r* in Live mode**
   *(input: event log, process tree(s); output: live view of the ETM*r *algorithm constructing a Pareto front of (repaired) process trees)*

Each of the three ETM algorithms (ETM*d*, ETM*r* and ETM*c*) is available in three versions. The basic version produces a process tree and provides high-level progress information such as the current generation and the overall quality of the best process tree. The Pareto version produces a Pareto front of process trees while high-level progress information is provided. The live Pareto version shows a live view of the Pareto front while the ETM algorithm continues to run in the background and adds process trees to it.

We would like to repeat the discovery of a Pareto front of process trees us-

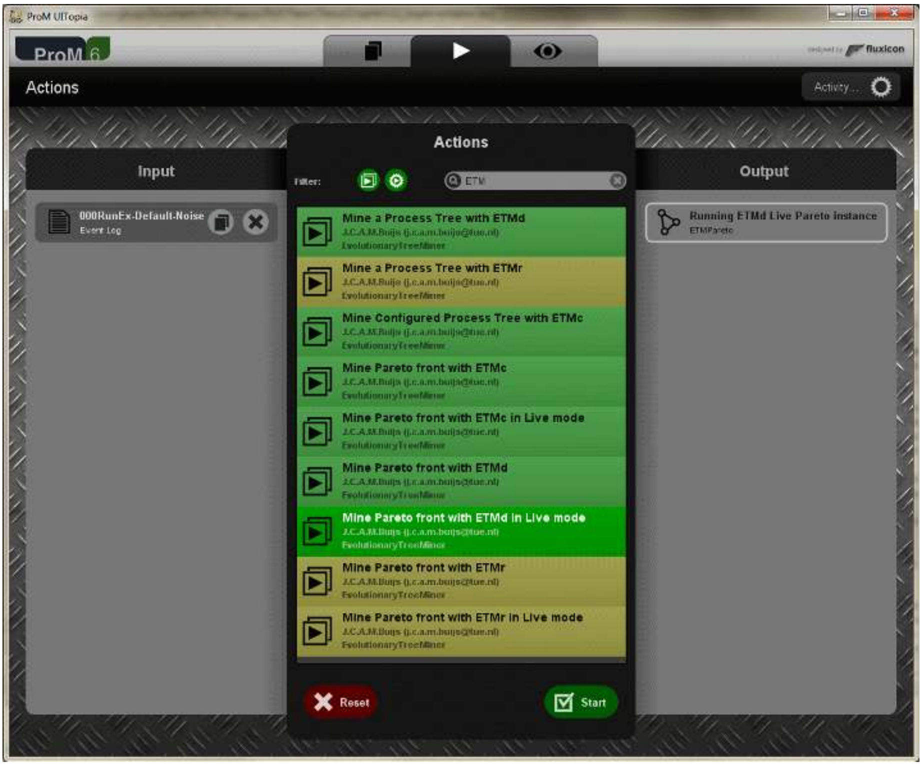

ProM UITopia
ProM 6
fluxicon
Actions
Activity...
Actions
Input
Filter:
ETM
Output
000RunEx-Default-Noise
Event Log
Mine a Process Tree with ETMd
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine a Process Tree with ETMr
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine Configured Process Tree with ETMc
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine Pareto front with ETMc
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine Pareto front with ETMc in Live mode
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine Pareto front with ETMd
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine Pareto front with ETMd in Live mode
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine Pareto front with ETMr
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Mine Pareto front with ETMr in Live mode
J.C.A.M.Buijs (j.c.a.m.buijs@tue.nl)
EvolutionaryTreeMiner
Running ETMd Live Pareto instance
ETMPareto
Reset
Start

Figure 11.1: Plug-ins provided by the ETM package in ProM.

ing the ETM*d* algorithm. Plug-ins 6 and 7 both produce a Pareto front using the ETM*d* algorithm. We select plugin 7 ('Mine Pareto front with ETM*d* in Live mode') since this provides detailed insights into the progress of the ETM*d* algorithm. All of the plug-ins present the same set of setting screens, which are shown in Figure 11.2. These setting screens help the user configure the required settings to run the selected ETM algorithm. All plug-ins utilize the same wizards with slight modifications depending on the algorithm type and version.

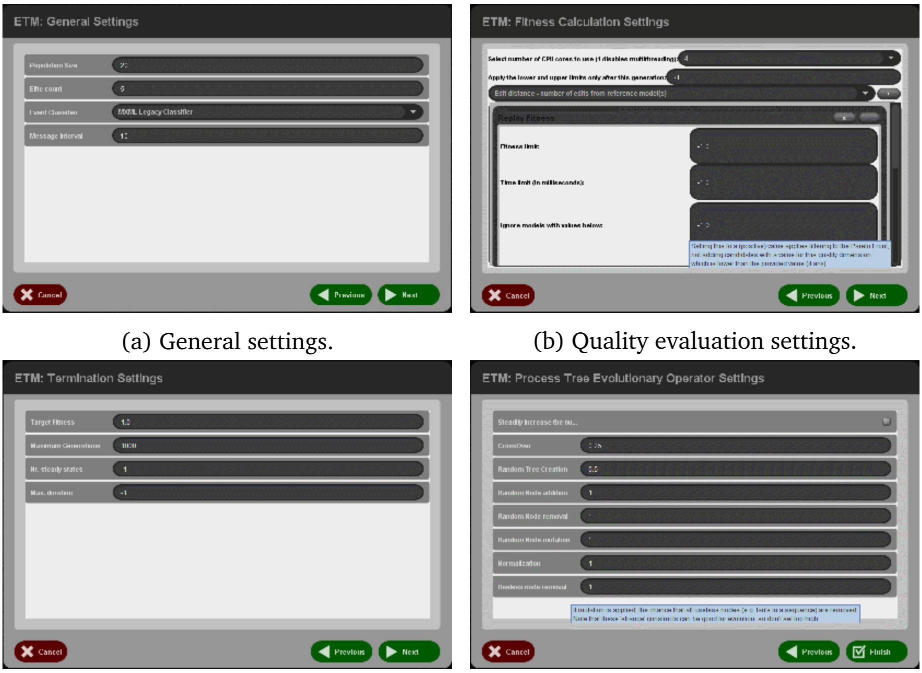
The first wizard screen, shown in Figure 11.2a, allows the user to define the population size and elite count. When the user hovers his mouse over the labels, an explanation is shown of the meaning of each setting together with value recommendations. The event classifier can be used to indicate how activities are defined. Note that this allows all ETM algorithms to also discover a social network when an activity is defined as the resource that executed an event. Finally, the message interval determines the interval at which progress is communicated.

In the quality calculation settings wizard, shown in Figure 11.2b, the user can specify which metrics should be used during evaluation. Note that different metrics may exist for a given quality dimension. The drop-down box at the top right allows the user to set the number of CPU cores to utilize during the evaluation. In case of a Pareto front discovery, the Pareto front can be pruned by providing upper and lower limits for each of the quality metrics. Candidates with a value outside of these bounds are not included in the Pareto front. However, since the quality of candidates in early generations might not be good enough, the second parameter allows the user to set the generation at which these limits are actually applied. The third line in the wizard allows the user to add all currently known quality metrics by selecting the metric from the drop-down box and then pressing the '+'-button. These quality metrics could be defined outside the ETM package, as is explained in Section 11.2.1. For each quality metric several parameters can be defined. For the replay fitness quality metric for instance the lower fitness limit can be provided. If, during calculation, it becomes clear that a process tree will have a quality value below this threshold, the calculation is aborted. Additionally, the second parameter allows the user to set how long the calculation per trace is allowed to take. Since both are set to -1 here these are not applied. Next the 'ignore models with values below' parameter is the limit applied on the Pareto front, as is explained by the blue explanation box shown. In case an algorithm is not run in Pareto front mode weights can be provided for each quality metric. These weights are used to calculate a weighted average over all quality metrics in order to obtain a single quality value per process tree.

The termination settings screen, shown in Figure 11.2c allows the user to specify several conditions which cause the ETM algorithms to terminate execution as soon as at least one of these conditions is met. Again, negative values indicate that the condition is disabled.

The last settings screen allows the user to specify the likelihood of different mutation operators being applied. Here again weights are used that indicate the relative chance of a mutation operator being applied. The blue box explains the last mutation operator which removes useless nodes.

The moment the user presses the 'Finish'-button in the wizard the ETM*d* algorithm is started and a Pareto front visualization is displayed, as shown in Figure 11.3. On the right-hand side of this visualization different navigators are presented that allow the user to inspect the Pareto front and the process trees contained in it. On the left-hand side of this view the currently selected process



ETM: General Settings

Population Size: 20
Elite count: 5
Event Classifier: MXML Legacy Classifier
Message Interval: 10

(a) General settings.

ETM: Fitness Calculation Settings

Select number of CPU cores to use (1 disables multithreading): 4
Apply the lower and upper limits only after this generation: -1
Edit distance - number of edits from reference model(s)

Replay Fitness
Fitness limit: -1.0
Time limit (in milliseconds): -1.0
Ignore models with values below: -1.0

Setting this to a (positive) value applies filtering to the Pareto front, not adding candidates with a value for this quality dimension which is lower than the provided value (-1 = none)

(b) Quality evaluation settings.

ETM: Termination Settings

Target Fitness: 1.0
Maximum Generations: 1000
Nr. steady states: 1
Max. duration: -1

(c) Termination settings.

ETM: Process Tree Evolutionary Operator Settings

Steadily increase the nu... □
CrossOver: 0.25
Random Tree Creation: 0.5
Random Node addition: 1
Random Node removal: 1
Random Node mutation: 1
Normalization: 1
Useless node removal: 1

If mutation is applied, the chance that all useless nodes (e.g. taus in a sequence) are removed. Note that these 'strange' constructs can be good for evolution, so don't set too high

(d) Change operator settings.

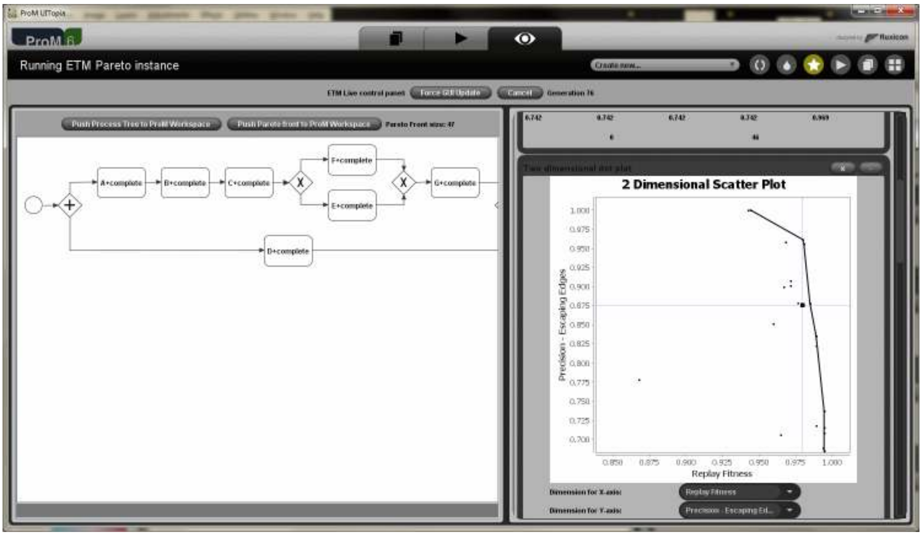
Figure 11.2: ETM parameter wizard screens.

tree is shown, in this case visualized as a BPMN model.

The currently selected process tree can be pushed to the ProM workspace using the ‘Push Process Tree to ProM Workspace’-button. This provides the process tree as an object in the ProM workspace so that it can be used for further analysis. In a similar way the currently discovered Pareto front can be made available in the ProM workspace. Next to these buttons the current size of the Pareto front is shown.

The ETM framework can be stopped by pressing the ‘Cancel’-button at the top. A forced update of the GUI of the current Pareto front can also be triggered, since updates can be delayed (currently the graphical user interface is set to update at most every 10 seconds). Next to these two buttons the currently visualized generation is shown.

At the right-hand side of the GUI different visualizations and Pareto front navigators are shown. Currently the ‘2 Dimensional Scatter Plot’-visualization is shown which projects the candidates of the Pareto front onto a two-dimensional scatter plot. The x-axis is currently set to show the replay-fitness quality metric, and the y-axis shows the precision (escaping edges) quality metric. The drop-



Dimension for X-axis: Replay Fitness

Dimension for Y-axis: Precision - Escaping Ed...

Figure 11.3: Result of the ‘Mine Pareto front with ETM*d* in Live mode’-plugin. The current process tree is shown on the left as a BPMN model. The 2d scatter plot Pareto front navigator is shown on the right.

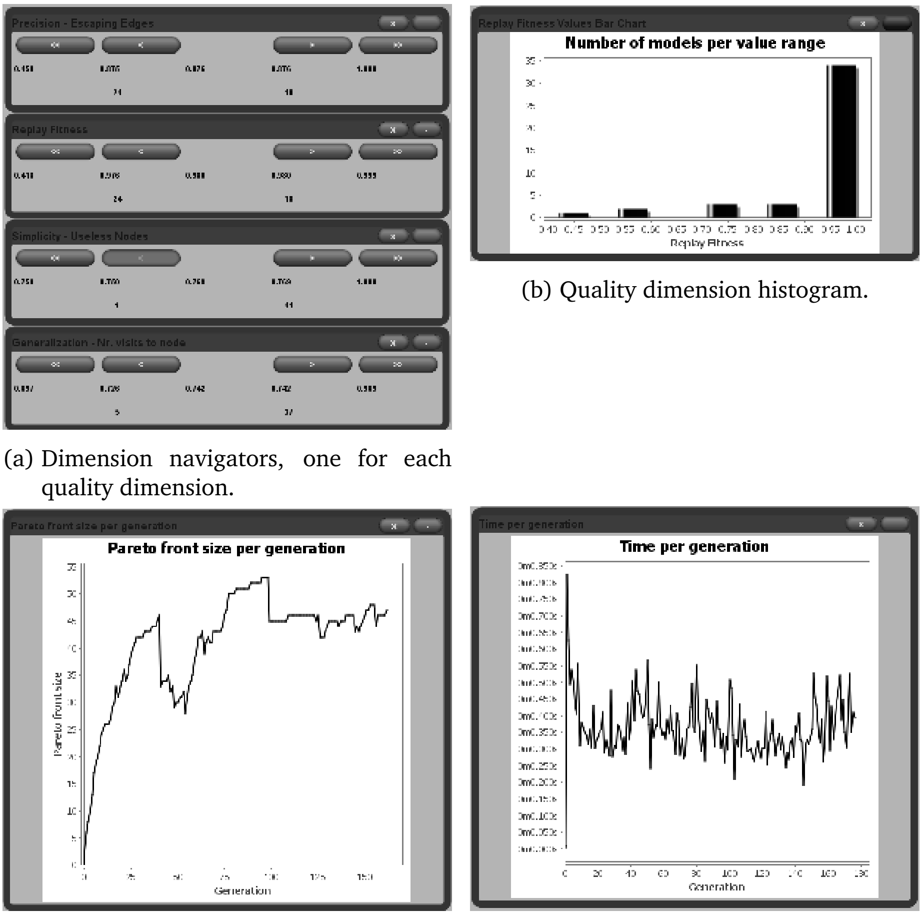
down boxes below the plot allow the user to freely change the quality metric shown on each of the axes. Each dot in this scatter plot is a process tree currently in the Pareto front. The chart is currently zoomed to show all candidates, and the user is free to zoom in and out. The process trees that together form a sub-front on the two selected dimensions are connected with a solid line. By clicking on the graph the process tree located closest to that location is selected and shown on the left. The currently selected process tree is indicated by the two crossing blue lines. This candidate is not on the sub-front of replay fitness and precision. In this example the currently selected process tree is also visualized as a square in the dot plot, instead of a dot. This indicates that this process tree should actually be removed from the Pareto front. This can happen if better process trees were found after the user selected this process tree. As soon as another process tree is selected using any of the visualizations, this process tree is removed from the Pareto front.

Several Pareto front visualizations and navigators are currently provided, and more can easily be implemented for (live) Pareto front visualizations. Figure 11.4 shows a selection of the other Pareto front visualizations that are currently provided.

Figure 11.4a shows the 'dimension navigators' for each of the quality metrics considered. Using the corresponding buttons, each dimension navigator allows the user to navigate to the worst candidate ('<<'), one candidate worse than the current process tree ('<'), one candidate better than the current process tree ('>'), and to the best candidate ('>>') for that quality metric. Below the buttons the values for the worst, one candidate worse, current, one candidate better and best candidate are shown. For the quality dimension of precision these values are 0.458 (worst), 0.875 (one worse), 0.876 (current), 0.876 (one better) and 1.000 (best). In the third row the number of candidates worse or better than the currently selected candidate are shown. For example, for precision there are 24 worse and 18 better candidates than the currently selected one.

Another visualization, which is also used in this thesis, is a histogram view, as shown in Figure 11.4b. This graph shows the distribution of the candidates in the Pareto front over the values of a quality metric. One histogram is shown for each of the quality metrics. The histogram of Figure 11.4b for instance shows the distribution of the candidates over the replay fitness quality dimension. Most candidates have a value between 0.95 and 1.0, while some have values as low as 0.4.

Since the Pareto front is still evolving in the live view, information over the generations can also be shown. The graph of Figure 11.4c for instance shows the number of candidates in the Pareto front for each of the generations. From this graph it can clearly be observed that the size increases quickly in early generations and is sometimes reduced in later generations. The time required per generation can also be visualized, as is shown in Figure 11.4d. This graph


Precision - Escaping Edges
Replay Fitness
Simplicity - Useless Nodes
Generalization - Nr. visits to node
Replay Fitness Values Bar Chart
Number of models per value range
Replay Fitness
Pareto front size per generation
Pareto front size
Generation
Time per generation
Generation

(a) Dimension navigators, one for each quality dimension.

(b) Quality dimension histogram.

(c) Graph showing the Pareto front size over the generations.

(d) Graph showing time per generation.

Figure 11.4: Different visualizations and navigators for the live Pareto front visualization.

shows that the time on average is stable, but that the first generation took more time.

As soon as one of the termination conditions is triggered, the ETM*d* algorithm will stop updating the Pareto front. The visualization continues to allow the user to further investigate the Pareto front, for instance by saving it to file, or by extracting process trees from it for further analysis.

11.1.2 Usage via code

The ETM framework can also be run via (Java) code. The minimal code required to perform the experiment of Section 7.1.3 is shown in Listing 11.1. The code first instantiates an event log. In this case we use a function to create the running example event log, but using the OpenXES library [176] event logs can also be loaded from file. Next the required parameters are initialized for the experiment. These include for instance the population size, chance of crossover and random mutation, as well as the maximum number of generations to run. Next an `ETMParamPareto` parameter object is instantiated using the event log and parameters. The parameter object contains all information and settings required by the ETM*d* algorithm to run. Most parameters have default settings, such as the selection mechanism, change operations and quality metrics used. Currently these are set to default values which can be used in most situations. The `ETMParamFactory` class also contains other methods that allow for more detailed instantiation of the parameter object. Furthermore, after instantiation of the parameter object all parameters can be inspected and changed, if desired.

With the parameter object correctly instantiated the ETM algorithm can be instantiated and run, in this case to discover a Pareto front. When the ETM algorithm is finished, the Pareto front can be obtained and processed further. In this example we first output the size of the discovered Pareto front and then save the Pareto front to file for further analysis.

11.1.3 Usage via Command Line Interface

A third option to run the ETM algorithms is by calling them via a command line interface (CLI). The main purpose of the command line interface is to allow ETM algorithms to be run on remote computers. It does not provide the usability and extensibility features offered via the graphical user interface or code interaction.

Calling ETM algorithms via the command line can be done by first calling the java application and providing it with the ETM code in the form of a JAR

Listing 11.1: Minimal example to the run ETM*d* algorithm from code.

```java
/**
 * Minimal code to run the ETMd to discover a Pareto front
 *
 * @param args
 */
public static void main(String[] args) throws IOException {
  //We instantiate one of our default event logs, an external event log can also be loaded.
  XLog eventlog = StandardLogs.createDefaultLogWithNoise();

  //Initialize all parameters:
  int popSize = 100; //population size
  int eliteSize = 20; //nr of trees in the elite
  int nrRandomTrees = 2; //nr of random trees to create each generation
  double crossOverChance = 0.1; //chance of applying crossover
  double chanceOfRandomMutation = 0.5; //chance of applying a random mutation operator
  boolean preventDuplicates = true; //prevent duplicate process trees within a population (after change operations are applied)
  int maxGen = 10000; //maximum number of generation to run
  double targetFitness = 1; //target fitness to stop at when reached
  double frWeight = 10; //weight for replay fitness
  double maxF = 0.6; //stop alignment calculation for trees with a value below 0.6 for replay fitness
  double maxFTime = 10; //allow a maximum of 10 seconds per trace alignment
  double peWeight = 5; //weight for precision
  double geWeight = 0.1; //weight for generalization
  double suWeight = 1; //weight for simplicity
  //the first null parameter is a ProM context, which does not need to be provided
  //the second null parameter is an array of seed process trees, which we do not provide here
  //the last ‘0’ is the similarity weight
  ETMParamPareto etmParam = ETMParamFactory.buildETMParamPareto(eventlog, null, popSize, eliteSize,
      nrRandomTrees, crossOverChance, chanceOfRandomMutation,
          preventDuplicates, maxGen, targetFitness,
      frWeight, maxF, maxFTime, peWeight, geWeight, suWeight, null, 0);

  ETMPareto etm = new ETMPareto(etmParam); //Instantiate the ETM algorithm
  etm.run(); //Now actually run the ETM algorithm, this might take a while

  //Extract the resulting Pareto front
  ParetoFront paretoFront = etm.getResult();

  System.out.println("We have discovered a Pareto front of size " + paretoFront.size()); //output the size
  ParetoFrontExport.export(paretoFront, new File("myParetoFront.PTPareto")); //and write to file
}
```

file. This is followed by the command line parameters which are parsed by the ETM algorithm. The call used to run the experiment of Section 7.1.3 via the command line interface is shown in Listing 11.2.

In this command, first Java is started and instructed to read a JAR file. Additionally the library path is set since the ETM requires an additional library[3]. Next the ETMCLI.jar is passed and parameter options are set. The first option is always the 'mode' in which the ETM algorithm should run, in this case Pareto mode. Other modes include 'NORMAL' (i.e., ETM*d*), and several specific modes for the four variants for discovering a configurable process tree as discussed in Chapter 8. The second option is always the output log directory, in which the intermediate and end results are written. The `log` parameter indicates which event log should be loaded. Next the weights for the four quality metrics are provided. The `maxGen` parameter indicates the maximum number of generations to run. Population and elite size are set next. The time limit (in seconds) for a single trace alignment by the replay fitness metric is set to 10 seconds. The random mutation ratio is set to 0.50. The `logmodulo` parameter indicates after how many generations a log file should be created. Since we set this parameter to 10, every 10 generations the current Pareto front is written to a file for analysis purposes. Additionally, a single statistics file is maintained with details of each generation.

All experiments performed in this thesis are executed via the command line interface. This allowed us to utilize a cluster of servers perform the experiments. The interested reader is referred to the `SSHExperiments` class in the `org.processmining.plugins.etm.experiments` Java package in the ETM source code[4]. This class automatically starts specified batches of experiments on a given set of servers. It also logs the console output to local files and starts new experiments when a server has finished its current experiment. In case an

[3] The LPSolve library is required, files for several operating systems can be found in the 'locallib' folder of the ETM source code.

[4] The initial version of the `SSHExperiments` class was implemented by Boudewijn van Dongen.

Listing 11.2: CLI call to start the experiment from Section 7.1.3.

```
java -jar -D"java.library.path=./lib/" ETMCLI.jar PARETO, /logDir/, log
    =/000RunEx-Default-Noise.xez, Fr=10, Pe=5, Sm=1, Gv=.1, maxGen
    =10000, popSize=100, eliteSize=20, limitFTime=10, randomMutRatio
    =0.50, logModulo=10
```

experiment failed, it is restarted. When all experiments for a particular experiment batch are completed, the log files of the servers are compressed and downloaded to the local log directory. If a next batch of experiments exists this batch is then started. The `org.processmining.plugins.etm.experiments.thesis` Java package in the ETM source code contains all experiment settings used in this thesis in the `ThesisExperimentSettings` class. Some of the experiment results can be processed by using functions in the `ThesisExperimentProcessor` class. Note that this last class contains experimental code and may not function 'out of the box'.

11.2 Extending the ETM*d* Algorithm

The ETM framework can easily be extended by adding more quality metrics, change operators and Pareto front visualizations. In this section we discuss in more detail how each of these can be included in the ETM algorithms.

The ETM*d* algorithm can be further extended by using new termination conditions, selection mechanisms, observer and logging classes and even with new engines and process tree extensions. The ETM package uses the Watchmaker framework [73] for evolutionary computing as the underlying framework, but extends (and modifies) it in many areas. The specific extensions can be included by following the guidelines for the Watchmaker framework [73].

11.2.1 Adding Quality Metrics

The main focus of the ETM framework is the incorporation of different quality metrics during process discovery. Therefore, it is very easy to implement new quality metrics and use them in the ETM framework. A quality metric should extend the abstract quality metric Java class `TreeFitnessAbstract`. This comes with the obligation to implement the `getFitness` function which, given a process tree and the current population, returns a numeric value. Additionally, a new quality metric can implement a graphical user interface that is able to configure that quality metric. This graphical user interface is shown in the ProM wizard for the user to configure the quality metric.

Any new quality metric should also provide a `TreeFitnessInfo` object which contains information such as a two-character code, name and description of the metric. It also contains the quality dimension the metric is related to, which is one the four common quality dimensions (replay fitness, precision, generalization and simplicity), a meta quality dimension that combines several other metrics, or 'other' to indicate that it measures another quality dimension. This information object also contains other quality metrics the current metric depends on. For instance, the metric that calculates precision by using escaping edges, as used in this thesis, depends on the replay fitness quality metric. Therefore, the replay fitness metric should be evaluated first, before the precision metric can be evaluated. Finally, the information object indicates whether bigger or smaller values are better according to this metric.

When the ETM plug-ins are run from within the ProM framework, all quality metrics are listed in the dropdown box shown at the top of Figure 11.2b. Via the ProM framework all known classes extending the `TreeFitnessAbstract` class are listed in this dropdown box. This means that authors can implement quality metrics in their own packages and the ETM framework is able to find them and present them to the user via the graphical user interface.

If the ETM is run via code, then the new quality metric can easily be added in the parameter object to use. Currently it is not possible to use arbitrary quality metrics via the command line interface without modifying the ETM source code.

11.2.2 Change Operations and Process Tree Creation

The performance of the ETM algorithms can be improved by including smarter operations to create and change process trees. New process tree creation mechanisms should extend the `TreeFactoryAbstract` class and implement a function that, given an event log, produces a process tree. In a similar way, classes providing new mutation operators should extend the `TreeMutationAbstract` class and return a new process tree from a given process tree. New crossover operations should extend the `AbstractCrossover<NAryTree>` class and should provide a function that, given two process trees, returns two modified process trees. These operations have access to the event log, quality scores of the process tree(s) and the alignment of the process tree(s) with the event log.

New creation and change operations are currently not automatically detected in the graphical user interface or the command line interface. Therefore, these can currently only be used by adding them to the parameter object via code or by modifying the code for the graphical user interface and command line. We plan to implement automatic detection of these operators for the graphical user interface, as exists for the quality metrics, before the official release of the code.

11.2.3 Pareto Front Visualizers

For the visualization of the Pareto front many visualizers and navigators are possible. Currently several interactive visualizers are implemented which are shown in Figure 11.4. New visualizers can be added to the Pareto front visualization as long as these new classes extend `AbstractParetoFrontNavigator`.

This visualization is notified when the currently selected process tree is updated. The visualizer itself has access to the Pareto front and its visualization and can therefore also update the selected process tree. This keeps all visualizers synchronized.

11.3 Implementation of the Comparison Framework

The comparison framework has been implemented as a plug-in in the ProM framework [176] in the `ComparisonFramework` package. This package is available as of ProM 6.4 and is currently included in the ProM nightly build[5]. The comparison framework package provides several plug-in variants that take one or more event logs, and one or more process trees.

The comparison framework visualization that is shown when the plug-in is started is shown in Figure 11.5. It consists of two main areas: the comparison table on the left and the settings on the right. In the figure the event logs and models from the case study of Section 10.4 are loaded. The event log metric selected is the average trace duration in days. The current process model metric simply displays the number of nodes in the process tree. Replay fitness is currently selected as the comparison metric, and replicates Figure 10.3a.

11.3.1 Metric Settings

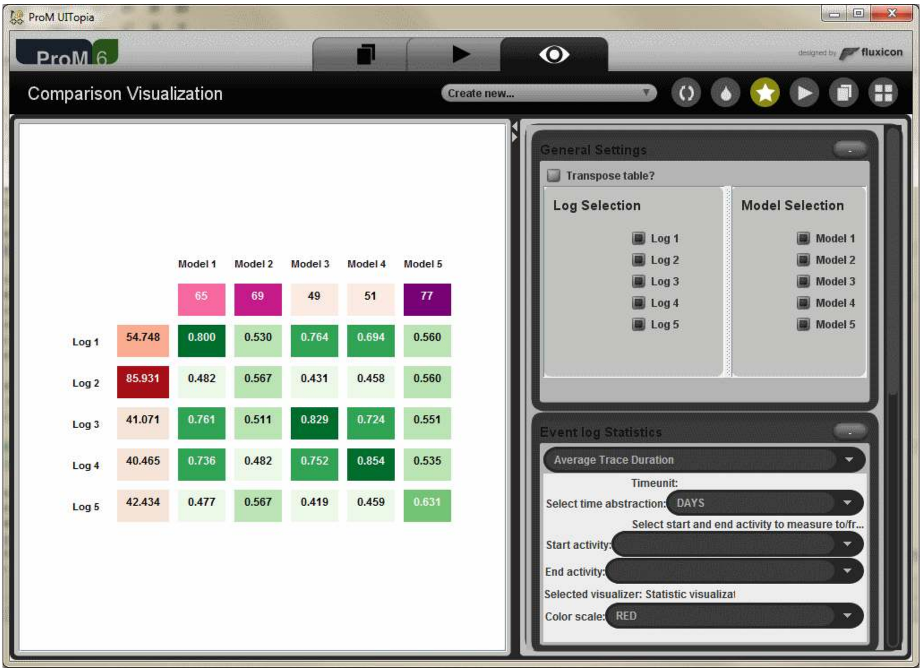
The settings panel of the comparison framework consists of several sections, which are shown in Figure 11.6. The top panel, shown in Figure 11.6a, allows the user to select which event logs and process models to show in the comparison table. By default event logs are rows and process models are columns, but this can be swapped by selecting the 'transpose table'-checkbox.

Next the event log, process model and comparison metric can be selected in their respective panels. Each metric produces an object of a certain type (or more specifically, a certain Java class). These objects are stored in a cache so time consuming calculations only need to be performed the first time the

[5] ProM 6 nightly can be obtained from `http://www.promtools.org/prom6/nightly/`.

metric is selected. In essence, we follow a model-view-controller approach [87] where metrics are the controllers that create objects. Since each object type can have multiple visualizers, the visualizers provide different views of the created object. Numbers for instance can be visualized by just showing the number, but also by coloring the cell background using different color scales, as is shown in Figure 11.5.

We shall not go into the details of the settings for all metrics and visualizers, but we choose to highlight some. Consider for instance the event log metric 'average trace duration', for which the settings panel is shown in Figure 11.6b. The time unit or time abstraction can be selected from nanoseconds, seconds, days, weeks, months and years. Furthermore, the activities can be selected between which the duration should be calculated. In case no activity is selected, the first or last activity in the trace is used to determine the start or end time of the trace respectively. This metric results in a statistics object, which can



		Model 1	Model 2	Model 3	Model 4	Model 5
		65	69	49	51	77
Log 1	54.748	0.800	0.530	0.764	0.694	0.560
Log 2	85.931	0.482	0.567	0.431	0.458	0.560
Log 3	41.071	0.761	0.511	0.829	0.724	0.551
Log 4	40.465	0.736	0.482	0.752	0.854	0.535
Log 5	42.434	0.477	0.567	0.419	0.459	0.631

General Settings

□ Transpose table?

Log Selection: ■ Log 1 ■ Log 2 ■ Log 3 ■ Log 4 ■ Log 5

Model Selection: ■ Model 1 ■ Model 2 ■ Model 3 ■ Model 4 ■ Model 5

Event log Statistics

Average Trace Duration

Timeunit:

Select time abstraction: DAYS

Select start and end activity to measure to/fr...

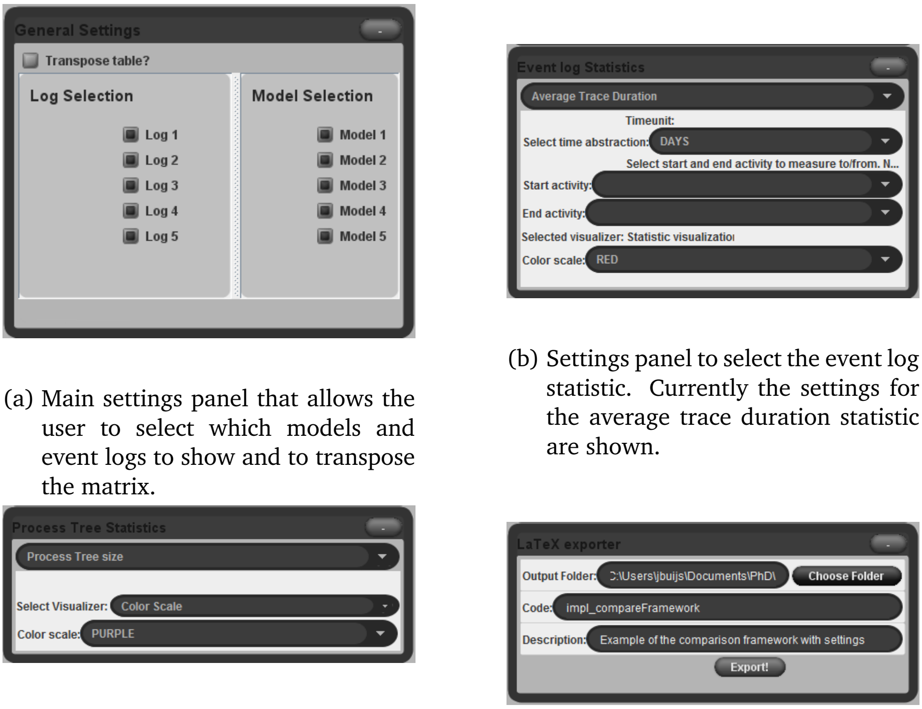
Start activity:

End activity:

Selected visualizer: Statistic visualizat

Color scale: RED

Figure 11.5: Main user interface of the comparison framework.



General Settings

□ Transpose table?

Log Selection: ■ Log 1, ■ Log 2, ■ Log 3, ■ Log 4, ■ Log 5

Model Selection: ■ Model 1, ■ Model 2, ■ Model 3, ■ Model 4, ■ Model 5

(a) Main settings panel that allows the user to select which models and event logs to show and to transpose the matrix.

Event log Statistics

Average Trace Duration

Timeunit:

Select time abstraction: DAYS

Select start and end activity to measure to/from. N...

Start activity:

End activity:

Selected visualizer: Statistic visualizatio

Color scale: RED

(b) Settings panel to select the event log statistic. Currently the settings for the average trace duration statistic are shown.

Process Tree Statistics

Process Tree size

Select Visualizer: Color Scale

Color scale: PURPLE

(c) Settings panel to select the process tree statistic. Currently the process tree size statistic is shown with the color scale visualizer chosen and set to the purple color scale.

LaTeX exporter

Output Folder: C:\Users\jbuijs\Documents\PhD\

Code: impl_compareFramework

Description: Example of the comparison framework with settings

(d) Additional feature to export the current comparison framework view to a LaTeX file and separate image files.

Figure 11.6: Overview of the main, event log and model statistic settings panel as well as the LaTeX export settings panel.

currently only be visualized by displaying the mean as a number with a colored background.

The settings panel for the process tree metric is shown in Figure 11.6c. In this case the metric only shows the size and has no additional parameters. Again, the number can be visualized using a color scale which can be selected.

In order to make the presentation of the comparison framework in case studies, papers and PhD theses easier, export functionality to LaTeX is also implemented, as is shown in Figure 11.6d. The output folder can be specified for the LaTeX file that represents the currently visualized comparison table. Each cell of the LaTeX table consists of text or includes a graphic that is also exported. Both the LaTeX file and the graphics have a file name that is prefixed with the 'code' field, which is also used to label the table. Finally, the caption of the table can already be filled in in the 'description' field.

11.3.2 Alignment Matrix Settings

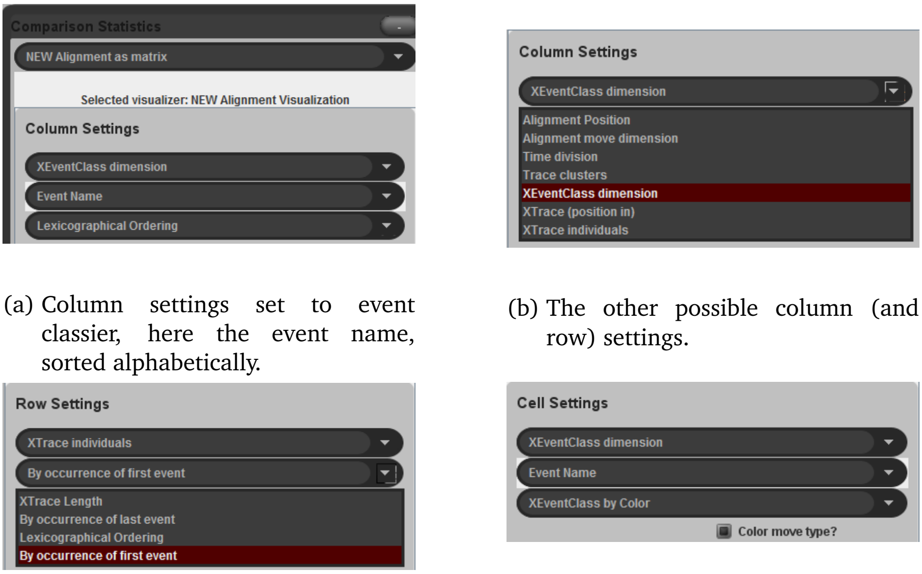
We explain the settings of the alignment matrix, which is discussed in Section 10.3, as a comparison metric. The alignment matrix again consists of columns, rows and cells that can be colored. In order to reproduce the alignment matrices used in Chapter 10, we require the following settings, which are also shown in Figure 11.7. The columns are set to represent the values of an event classifier (see [176]), as is shown in Figure 11.7a. The event classifier in turn is set to the activity name, but resources can also be specified. Now, each column represents an activity and the columns are sorted alphabetically. Figure 11.7b shows the other representation options for columns and rows. These can be set to represent traces, time periods or the position in the trace or alignment. The rows, as shown in Figure 11.7c, are currently set to represent traces, sorted chronologically on the first event (i.e., oldest traces first). The color of each cell is set to represent the activity or move type, as shown in Figure 11.7d. This means that a cell is white (no move), gray (model moves), black (log moves) or 'colored' where each activity is assigned a unique color. In case the 'Color move type?'-checkbox is not checked, all cells are colored according to the activity of the model move or log move, i.e., there are no black or gray cells. The many setting combinations allow the alignment matrix to visualize many different comparisons between an event log and a process model.

11.3.3 Extending the Comparison Framework

The comparison framework is set up in a flexible way so new metrics and visualizers can easily be added. Other packages can provide new event log metrics, process tree metrics, comparison metrics or visualizations of objects. These are automatically included in the user interface of the comparison framework.

Metrics need to extend the corresponding abstract metric class. The extending classes should provide the class of their return type (e.g. number, alignment matrix or a Java graphical element). Additionally they should implement a function that provides an instance of this class, based on the provided event log and/or process tree.

Visualizers should indicate which Java class they are able to visualize. In case a subclass is produced, visualizers of superclasses are also included and



Comparison Statistics

NEW Alignment as matrix

Selected visualizer: NEW Alignment Visualization

Column Settings

XEventClass dimension

Event Name

Lexicographical Ordering

(a) Column settings set to event classifier, here the event name, sorted alphabetically.

Column Settings

XEventClass dimension

Alignment Position
Alignment move dimension
Time division
Trace clusters
XEventClass dimension
XTrace (position in)
XTrace individuals

(b) The other possible column (and row) settings.

Row Settings

XTrace individuals

By occurrence of first event

XTrace Length
By occurrence of last event
Lexicographical Ordering
By occurrence of first event

(c) Row settings set to individual traces sorted by occurrence of first event.

Cell Settings

XEventClass dimension

Event Name

XEventClass by Color

■ Color move type?

(d) Cell settings set to event name by color, with special colors for the non-synchronous move types.

Figure 11.7: Overview of the alignment matrix comparison statistic, and its settings for the column, row and cell representation.

can be called. Additionally, visualizers can implement a specific function that can decide on a concrete object instance whether they can visualize the instance. Visualizers should of course produce a Java graphical element (more concretely a `JComponent` or subclasses thereof) when objects of the declared type are provided.

Both metrics and visualizers can provide a settings panel for the user to configure the calculation and visualization.

11.4 Conclusion

In this chapter we discussed how the ETM framework and the ETM algorithms can be used. We have shown how one of the experiments performed in this thesis can be reproduced using three approaches. The graphical user interface of ProM provides the most user friendly interaction. Java code calls allow for the most flexibility and parameter tuning. The command line interface is of use when many experiments are to be performed automatically, but this approach provides less parameter options.

Additionally, in this chapter we have discussed several ways in which new elements, such as quality metrics or change operators, can be added to the ETM framework for use by all ETM algorithms.

In this chapter we have also discussed the implementation and use of the comparison framework as presented in Chapter 10. The currently implemented statistics were discussed, with emphasis on the alignment matrix. Additionally it was discussed how new metrics and visualizations can be added to the comparison framework.

Chapter 12

Conclusion

In this chapter we summarize our main findings. In Section 12.1 we highlight the contributions and results of this thesis. Section 12.2 lists some of the current challenges and open issues that remain. Finally, in Section 12.3 we present an outlook for further research and some general issues that should be addressed in the general area of process mining.

12.1 Contributions of this Thesis

In the introduction of this thesis we presented seven challenges related to process discovery, process model repair and the analysis of process families. These seven challenges are:

1. Produce correct process models;
2. Separate visualization and the representational bias;
3. Balance the quality of discovered process models;
4. Improve understandability for non-experts;
5. Use existing knowledge in process discovery;
6. Describe a family of processes;
7. Compare similar observed behavior.

In Chapter 3 we addressed Challenges 1 and 2. We first discussed several requirements for process modeling notations. One of the main requirements is that a process model should be sound, i.e., error free, which is presented as Challenge 1. Common process modeling notations were then discussed and evaluated using the requirements identified. This showed that none of the existing process modeling notations can guarantee soundness without imposing severe restrictions. Therefore, we presented *process trees* as a new process modeling notation. Process trees are inherently sound because of their block structure. Additionally, we have shown that process trees can easily be translated to sound and well-structured process models in several different notations. As a result, the process tree notation also addresses Challenge 2 since it can be visualized using most of the common process modeling notations. We also discussed when and, if possible, how existing process models can be translated to process trees.

The Evolutionary Tree Miner (ETM) framework was presented in Chapter 4. This framework allows for flexible process discovery using evolutionary algorithms and is required to address the other challenges. After an introduction of the ETM framework several application scenarios were discussed where a flexible process discovery framework was required. The requirements for the different phases of the ETM framework were also discussed in detail. The main aspect to consider during implementation of the ETM framework is the balance between exploring all possibilities in the search space and quickly converging to optimal solutions. We also presented several common approaches from the field of evolutionary computing for the different phases of the ETM framework.

The relationships between observed behavior as recorded in the event log, the behavior of process models and the behavior of an unknown system were discussed in detail in Chapter 5. This is necessary to address Challenges 3 and 4, since balancing quality and improving understandability of process models heavily rely on a good understanding of the quality of a process model. The four well-known quality dimensions of replay fitness, precision, generalization and simplicity were considered in this discussion. Furthermore, for each of these four quality dimensions a metric was proposed. These are used to evaluate the quality in that dimension for a given process tree and event log. Several other metrics in each dimension were discussed and additionally we showed that all four quality dimensions should be considered in order to obtain meaningful and useful process models.

The first flexible evolutionary algorithm based on the ETM framework, the ETM*d* algorithm for process discovery, was presented in Chapter 6. The ETM*d* algorithm addresses Challenge 3 by considering all four quality dimensions during discovery. The implementation of all aspects of the ETM framework was discussed. These implementation efforts first resulted in the ETM*d* algorithm for process discovery. The ETM*d* algorithm is a flexible process discovery algorithm that is able to balance the quality dimensions discussed in Chapter 3. In Chapter 6 the ETM*d* algorithm was applied to an event log to show the discovered process trees. The discovered process trees were also compared with the results of existing process discovery techniques.

The applicability of the ETM*d* algorithm was further demonstrated in Chapter 7 where more extensive experiments were performed. The ETM*d* algorithm was applied to several artificial and real life datasets. We showed that the resulting process models are of equal or better quality than those created by existing process discovery algorithms. We also demonstrated, by presenting the discovered Pareto front of process trees, that there is no single process model that describes the observed behavior the best. Different aspects of the behavior of the ETM*d* algorithm were discussed such as the long-term behavior and the effects of random versus guided change operations. Since evolutionary algorithms are applicable to a wide range of problems, they are in general slower than problem tailored methods. Therefore, we investigated the performance of the ETM*d* algorithm in detail and concluded that the performance mainly depends on the calculation of the alignments.

In Chapter 8 we presented the ETM*r* algorithm which considers a normative process model during discovery, thus addressing Challenge 5. We showed that the only change required to transform the ETM*d* algorithm into the ETM*r* algorithm is the addition of similarity as a fifth quality dimension. By applying the ETM*r* algorithm on both a running and real life event log we demonstrated that the resulting process models indeed provide a good balance between the provided normative model(s) and the observed behavior.

In Chapter 9 we presented the ETM*c* algorithm which is able to discover a configurable process model, describing a collection of event logs, which addresses Challenge 6. After discussing configurable process models in general we discussed how the process tree notation was extended to capture configurations. We then introduced a way to change configurations in a configurable process tree and how to measure the configuration quality of a configurable process tree. We compared the ETM*c* algorithm with three other approaches for the discovery of a configurable process model proposed in literature. However, the ETM*c* algorithm has a clear advantage over these other approaches, for instance because it can construct a Pareto front of configurable process trees. We demonstrated this by comparing all four approaches on both a running example and real event logs.

Chapter 10 presented a comparison framework to compare inter-organizational processes, thus addressing Challenge 7. Within this framework we support the visualization of alignments between observed and modeled behavior without showing the process model. We have evaluated the framework, and mainly the alignment visualization, in a case study with partners from the CoSeLoG project.

The implementation of the ETM framework and the ETM algorithms presented in this thesis were discussed in more detail in Chapter 11. The ETM framework is implemented in the ETM package in the ProM framework and is included as of ProM 6.4. We demonstrated how one of the experiments performed in this thesis can be replicated via the graphical user interface of ProM, via Java code calls and via a command line interface. We also discussed how new elements, such as quality metrics and change operations, can easily be added to the ETM framework. The ProM framework and the ETM package can be obtained from `www.processmining.org`.

12.2 Current Challenges and Open Issues

In this section we discuss some of the limitations of the techniques presented in this thesis. We also present improvements that can still be made and discuss opportunities that we see based on extensions of the techniques presented.

12.2.1 Limitations

The work presented in this thesis provides a comprehensive approach to process mining and answers novel questions. However, the current approach and implementation also suffers from some limitations. Most notable improvement opportunities are:

Performance Improvement of Alignment Calculations. The performance of the ETM framework is currently mostly limited by the alignment calculations. Great effort has already been put into optimizing the alignment calculations for process trees. Also, for the ETM framework, in some cases estimations might be sufficient. Furthermore, given a process tree, its alignment and the change applied on this process tree, the new alignment could be deduced instead of calculated from the beginning.

Smarter Guided Mutation. In order to improve the performance of the ETM framework and the quality of the produced process models, more guided mutations should be implemented. If higher quality process models are created quicker, less alignments need to be calculated, so the performance of the ETM improves significantly. Additionally, smarter selection of which guided mutation to apply on a particular process tree results in better performance of the ETM algorithms and better quality of the discovered process trees. For instance, using the current quality of a process model, the guided mutation can be selected that is likely to improve a particular quality aspect of the process model the most.

Furthermore, guided mutations should not be restricted to control flow only, since guided repair and configuration mutations can also be created. In [141] a guided configuration mutation is discussed. This makes the search more effective by considering the alignments of each of the given event logs on the process model, in order to detect a suitable configuration.

Better Precision Metric. Currently the best metric for precision is the approach of [138] which uses the 'escaping' (i.e., not-used) edges in the state space that is created during the alignment calculations. When operator nodes are added that have only one child, the ratio of escaping edges is reduced, and thus precision is improved. These operator nodes are useless according to the definition of Section 5.3.1. The generalization metric proposed in this thesis already ignores useless nodes, and precision should do the same. Another issue related to the current precision metric is that the fraction of escaping edges is an estimation of the non-observed but modeled behavior, since the modeled behavior might be infinite. A single escaping edge, which is not investigated further, might therefore hide a lot of behavior that is currently not considered. This is reflected by the mild punishment for loops in process trees. A loop with children a, b and c and an event log consisting only of the trace $\langle a, c \rangle$ currently has a calculated precision of 0.667. This score does not express the imprecision of the loop operator well given the observed behavior.

12.2.2 Improvements

The work in this thesis can be further improved and extended by applying the following ideas:

Further Theoretical Discussion. In this thesis we discussed the relationships between the event log, process model and system. This theoretical discussion can be further investigated to thoroughly understand the quality dimensions and their interactions. For instance, currently it is unclear how noise and completeness can be measured. The main cause for this is that they reason about the behavior of the unknown system. Furthermore, there is no common understanding and agreement on the exact meaning and interpretation of generalization, noise and completeness. Some argue that noise should be removed from the event log before discovery is applied, while we and others see noise as infrequent or exceptional behavior that cannot be detected before process discovery is started.

New Generalization Metrics. In this thesis we proposed a generalization metric that follows our understanding of the generalization quality dimension. Few generalization metrics have been proposed in literature. We do not claim that our proposal is the definite and only metric to express this quality dimension. Although it seems to work well in the context of the ETM framework and ETM algorithms, other metrics might better capture the philosophy of the dimension. At the same time the generalization quality dimension is hard to measure since it reasons about an unknown process. However, by further reasoning about the quality dimensions, and the interplay between the process model, event log and the real process, new ideas for generalization metrics might be obtained.

Visualization of Quality Metrics on Process Trees. Currently the quality of a process tree is indicated by a numeric value for each of the quality dimensions. However, more detailed information is available. Alignments for instance can be projected onto process trees and process models in general [21]. However, precision, generalization and simplicity should also be projected onto the process tree. Furthermore, information such as visit frequencies and time-related information should also be visualized on a process tree to give insights into the quality of the process tree. A first approach has recently been presented in [184] but we foresee more ways of visualizing these properties.

Extend Expressiveness of Process Trees. Although process trees are able to express the five basic control-flow constructs (sequence, exclusive choice, parallelism, non-exclusive choice and loops), additional operators could be added. For instance, a 'long-term dependency'(LTD)-operator would be able to synchronize choices made in one part of the process with the choice to be made in another part. An example is shown in Figure 12.1. An LTD-operator would take three children, of which the first and the last are ×-operators with the same number of children. The choice made at the first ×-operator is synchronized with the last ×-operator, which activates the subtree at the same index. For the example of Figure 12.1, subtree X is executed if subtree A was executed before, and subtree Y is executed if subtree B was executed before. Note that long-term dependencies could also be encoded when data is included in the process discovery phase. Other operators that could be added to process trees are the milestone construct [17], cancelation regions [17] and interleaved routing [17].

12.2.3 Opportunities

The work presented in this thesis provides many opportunities for future work, some of which are discussed here.

Testing new Quality Metrics using the ETM Framework. One of the research purposes of the ETM framework is to investigate the quality dimensions and proposed quality metrics. New quality metrics can easily be added, and their effects on the resulting process models can be investigated. The ETM framework applies evolutionary techniques to optimize each of the quality dimensions provided. By inspecting whether the resulting process models correspond with the intention of the quality metric, these metrics can be improved. For instance, during early experiments using the ETM*d* algorithm we found process trees that scored very well on generalization, but did not correspond to our notion of a generalizing process tree. This was mainly caused by the ETM*d* algorithm adding many nodes to the process tree that were frequently visited but did not change the behavior. By ignoring these useless nodes in the generalization calculation the resulting process trees improved.

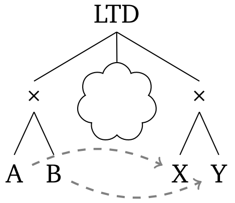


Figure 12.1: Example of a 'long-term dependency' (LTD) operator, which synchronizes the choices of the two ×-operators.

Structured Approach to Flexible Process Discovery. The ETM framework provides a framework for flexible process discovery. However, since the ETM framework applies evolutionary techniques, the approach is not structured. A more structured and constructive approach is necessary to be able to apply flexible process discovery on larger event logs within a reasonable execution time. The ETM framework allows for the addition of new tree creation, crossover and (guided) mutation change operations. By investigating which change operators work well together, a more structured and constructive approach can be discovered. This can eventually result in a structured approach that does not depend on the ETM framework any more. It is however a challenge to combine this with flexible discovery.

Use Available Information During Discovery. The event log contains more information than only the order in which activities are executed. Each event for instance contains information about the lifecycle state of the activity, i.e., if the activity was started, completed, canceled, paused, resumed, and so on. Using this information during mutation and evaluation can help in creating better process trees. It is for instance known that an activity (usually) can only be completed once it has started. This restricts the possible combinations of the activities in the process tree. Other knowledge such as activity duration, resource allocation, and recorded data values can also be used. All this information can help in deducing relationships between activities and thus help in providing better guided mutations.

Make ETM Algorithm Execution Interactive. During the execution of the ETM algorithms more interaction with the user can help focus efforts of the ETM algorithms. This can increase the quality of the discovered process trees or Pareto front by incorporating the user's requirements. The user can for instance indicate the area in the Pareto front they are interested in so the ETM algorithm can focus its efforts creating better and more process trees in that area. The user could also introduce additional quality dimensions, or remove existing quality dimensions that do not contribute to good quality process trees, at run-time.

Extended Usage of Process Trees. Process trees provide a unique structure to process models that can be used for other purposes than process discovery. For instance, calculation of alignments for the replay fitness dimension is more efficient on process trees than on Petri nets because of the structure of process trees. The process tree structure can however be used for more analysis and visualization techniques. A process tree can for instance give insights into the hierarchical grouping of activities. This can help in clustering activities in order to ensure that all activities are on the same level of abstraction. Another use case would be to support business process modeling by using a process tree. The user could directly model a process tree, or model in their preferred process modeling notation while a process tree is constructed in the background. This results in sound and structured process models.

Integration of the Comparison Framework into the Dashboard of the Cloud IS. Within the CoSeLoG project we envision a cloud-based information system [162] that supports the execution of similar processes across organizations (see Section 1.4). The techniques presented in this thesis, but especially the discovery of a configurable process model presented in Chapter 9 and the comparison framework presented in Chapter 10, can be integrated into the dashboard of this cloud system. The configurable proces model discovery can be integrated into the dashboard to provide up-to-date insights into suitable configurations based on the observed behavior. Incorporating the comparison framework into the dashboard enables a live and interactive comparison between the recorded behavior and process models of the different organizations. By investigating the process configurations of other organizations, and comparing these with the process execution characteristics such as costs and processing time, organizations can quickly learn and improve. Furthermore, new collaborations can be triggered on a daily basis.

12.3 Outlook on Process Mining

When we look beyond the scope of the work presented in this thesis, we observe that process mining consists of several areas, of which process discovery is often seen as the most important. In the other areas analysis techniques such as alignments between process models and event logs, visual analytics of event logs, and online process mining exist. In general these techniques are mature and robust enough to be applied on real life cases. However, process discovery is not that mature and robust yet. This can be explained by the observation that process discovery is much more difficult to grasp than visualizing observed events or predicting a next event. In this section we propose several aspects that should be addressed to improve the quality of process discovery algorithms, and process mining research in general.

12.3.1 Closer Collaboration between Academia, Tool Vendors, Consultants and Clients

There are several parties involved in developing and applying process mining techniques. Researchers in academia provide solutions to selected problems related to process mining. Some of these solutions are picked-up by tool vendors which incorporate similar solutions in their commercial (process mining) analysis tools. The tools, or sometimes even the academic solutions themselves, are used by consultants to answer questions they are asked to solve for their clients.

For all parties involved it is crucial that each party communicates with the other parties. In order to be able to answer questions that are asked by clients, consultants need appropriate tools. In essence, most of these tools can only be provided by the tool vendors if they have been researched and developed by researchers in academia. However, although researchers are very capable of finding new challenges to solve, they should be aware of the most pressing questions in industry.

Therefore, there should be feedback loops in order for current issues of clients to be collected by consultants and tool vendors so they reach researchers in academia. Of course, the results of researchers should be evaluated on client data, possibly via tool vendors and consultants, to validate the proposed solutions. In the end the researched solutions can be picked up by tool vendors, and used by consultants. However, researchers should not ignore the fact that consultants might use their solutions directly, and the solutions should be suitable for such usage.

12.3.2 Address Hindering Side Issues

There are several ‘side issues’ that are hindering the development and adoption of process mining as a whole. These issues involve privacy, transparency, lack of a clear approach or set of tools, and immature process discovery algorithms.

Within the CoSeLoG project we experienced a fruitful collaboration between the clients (the municipalities) and the involved tool vendors. Using the concrete questions but more importantly the concrete data of the municipalities, we found interesting questions to address and we were able to test our proposed solutions. It is however crucial that clients are willing to share their data with researchers in academia. Many colleague researchers have difficulties obtaining data because of lack of trust or privacy concerns from the clients. Research from a more privacy and security oriented perspective is necessary to address these concerns, probably as part of the whole ‘big data’ discussion. Only if privacy and related issues are addressed, closer collaborations between academia and clients become possible. Ideally there should be a standard agreement on how data is obtained, processed, stored and shared, as well as on how results are presented in publications.

12.3.3 Improve the Applicability of Solutions on Real Data

One of the main results from the proposed increased collaboration is the applicability of solutions proposed by academia on real data. Currently most proposed techniques and solutions are mainly validated on artificial data, sometimes with an evaluation on a specific real life data set included. The problem this causes is demonstrated by the low quality of the results of existing process discovery algorithms on the real life data used in this thesis (see Section 7.3). Although all process mining solutions should be applicable to real data, for process discovery algorithms this is crucial.

In the early days of process discovery simple techniques, such as the α-algorithm, were proposed. These algorithms were developed to discover process models while making strong assumptions about the process and event data. Furthermore, the whole problem of process discovery required further investigation and understanding which these simple algorithms helped achieve. However, currently many process discovery techniques exist and our understanding of the process discovery challenge has improved. Most of the current process discovery techniques have trouble addressing all challenges discussed in this thesis: producing correct results, providing insights into what makes a process model good, and providing alternative solutions. The ETM framework and ETM algorithms are not the definitive answer to these challenges. The ETM framework is a first implementation of what will hopefully become a new generation of process discovery algorithms that will actually produce useful results.

Developing good process discovery algorithms is also crucial for process mining as a whole. The discovered process models enable further analysis via a large collection of techniques. However, the quality of the process model, amongst several other characteristics, has a big influence on the quality (and therefore usefulness) of the subsequent analysis results.

12.3.4 Improve the Usability of Techniques

Many interesting process mining techniques and approaches have been proposed by researchers all over the world. Usability of most of these process mining techniques however remains an issue. Most techniques present the user with many parameters. Although these settings are often very clear and understandable for the author(s), other researchers have little knowledge of the inner workings of these proposed techniques. Therefore, if techniques are to be used by other researchers, or even by consultants in industry, there should at the very least be a simple set of parameters to specify. Explanations should be provided of the purpose and recommended values of these parameters, preferably in both the graphical user interface and code documentation. If the parameters are unclear, and researchers obtain bad results in experiments caused by incorrect parameter settings, adoption of a new technique will be hindered.

12.3.5 Create Incentives for Researchers to Publish and Document their Solutions

Although proposed techniques and solutions should be user-friendly, documentation is always necessary. But more crucially, before new techniques can be used and extended, they have to be made public. Currently however there is little incentive for researchers to publish their solutions and provide good quality documentation. Especially new researchers and consultants, who often have great interest in process mining, have little idea where to start. Providing a simple explanation of the basic usage of the different plug-ins which are for instance available in the ProM framework would help all users select the correct techniques for their tasks. This also will inspire them to contribute new ideas and techniques. One approach would be to make a specific code version, including documentation, into a publication that can be cited, and thus would contribute to an author‘s h-index. However, currently code and documentation are not regarded as quality publications so researchers have no reason to put effort into these.

Bibliography

[1] W.M.P. van der Aalst. Three good reasons for using a petri-net-based workflow management system. In S. Navathe and T. Wakayama, editors, *Proceedings of the International Working Conference on Information and Process Integration in Enterprises (IPIC'96)*, pages 179–201. Camebridge, Massachusetts, Nov 1996. (Cited on page 39.)

[2] W.M.P. van der Aalst. Configurable services in the cloud: Supporting variability while enabling cross-organizational process mining. In R. Meersman, T.S. Dillon, and P. Herrero, editors, *OTM Conferences (1)*, volume 6426 of *Lecture Notes in Computer Science*, pages 8–25. Springer, 2010. ISBN 978-3-642-16933-5. (Cited on pages 1, 19, and 20.)

[3] W.M.P. van der Aalst. Business process configuration in the cloud: How to support and analyze multi-tenant processes? In G. Zavattaro, U. Schreier, and C. Pautasso, editors, *ECOWS*, pages 3–10. IEEE, 2011. ISBN 978-1-4577-1532-7. (Cited on pages 1 and 20.)

[4] W.M.P. van der Aalst. On the representational bias in process mining (keynote paper). In S. Reddy and S. Tata, editors, *Proceedings of the 20th Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE 2011)*, pages 2–7. IEEE Computer Society Press, Paris, 2011. (Cited on pages 35, 39, and 40.)

[5] W.M.P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011. ISBN 978-3-642-19344-

6. (Cited on pages 3, 4, 5, 27, 28, 31, 33, 36, 40, 41, 78, 98, 103, 104, 120, 122, and 223.)

[6] W.M.P. van der Aalst. What makes a good process model? - Lessons learned from process mining. *Software and System Modeling*, 11(4):557–569, 2012. ISSN 1619-1366. (Cited on page 35.)

[7] W.M.P. van der Aalst. Decomposing Petri nets for process mining: A generic approach. *Distributed and Parallel Databases*, 31(4):471–507, 2013. (Cited on pages 29 and 30.)

[8] W.M.P. van der Aalst. Mediating between modeled and observed behavior: The quest for the "right" process: Keynote. In R. Wieringa, S. Nurcan, C. Rolland, and J.L. Cavarero, editors, *RCIS*, pages 1–12. IEEE Computing Society, 2013. ISBN 978-1-4673-2912-5. (Cited on pages 98, 99, and 122.)

[9] W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Causal nets: A modeling language tailored towards process discovery. In J. Katoen and B. Koenig, editors, *CONCUR*, volume 6901, pages 28–42. 2011. ISBN 978-3-642-23216-9. (Cited on pages 12, 43, and 49.)

[10] W.M.P. van der Aalst, A. Adriansyah, and B.F. van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012. ISSN 1942-4795. (Cited on pages 98, 104, 109, 118, 122, and 123.)

[11] W.M.P. van der Aalst, A. Adriansyah, A.K.A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, R.P.J.C. Bose, P. van den Brand, R. Brandtjen, J.C.A.M. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B.F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D.R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C.W. Günther, A. Guzzo, P. Harmon, A.H.M. ter Hofstede, J. Hoogland, J.E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F.M. Maggi, D. Malerba, R.S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H.R.M. Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H.S. Pérez, R.S. Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K.D. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, H.M.W. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Wei-

dlich, T. Weijters, L. Wen, M. Westergaard, and M.T. Wynn. Process mining manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-28107-5. (Cited on pages 10, 11, 13, 14, 15, 16, 35, and 349.)

[12] W.M.P. van der Aalst, J.C.A.M. Buijs, and B.F. van Dongen. Towards improving the representational bias of process mining. In K. Aberer, E. Damiani, and T. Dillon, editors, *SIMPDA*, volume 116 of *Lecture Notes in Business Information Processing*, pages 39–54. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-34043-7. (Cited on page 349.)

[13] W.M.P. van der Aalst, J. Desel, and E. Kindler. On the semantics of EPCs: A vicious circle. In M. Nüttgens and F. Rump, editors, *EPK*, pages 71–79. Gesellschaft für Informatik, Bonn, Trier, Germany, November 2002. (Cited on pages 36, 38, 43, and 50.)

[14] W.M.P. van der Aalst, A. Dreiling, F. Gottschalk, M. Rosemann, and M.H. Jansen-Vullers. Configurable process models as a basis for reference modeling. In C. Bussler and A. Haller, editors, *Business Process Management Workshops*, volume 3812, pages 512–518. 2005. ISBN 3-540-32595-6. (Cited on page 224.)

[15] W.M.P. van der Aalst, K.M. van Hee, A.H. ter Hofstede, N. Sidorova, H. Verbeek, M. Voorhoeve, and M. Wynn. Soundness of workflow nets with reset arcs is undecidable! In *Proceedings of the International Workshop on Concurrency Methods Issues and Applications (CHINA'08)*, pages 57–72. Xidian University, 2008. (Cited on page 36.)

[16] W.M.P. van der Aalst, K.M. van Hee, A.H.M. ter Hofstede, N. Sidorova, H.M.W. Verbeek, M. Voorhoeve, and M. Wynn. Soundness of workflow nets: Classiffication, decidability, and analysis. *Formal Aspects of Computing*, 23(3):333–363, 2011. (Cited on pages 36 and 37.)

[17] W.M.P. van der Aalst, A.H.M. ter Hofstede, B. Kiepuszewski, and A. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, 2003. (Cited on pages 38, 41, 42, and 303.)

[18] 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 behav-

ior. In S. Dustdar, J.L. Fiadeiro, and A.P. Sheth, editors, *Business Process Management*, volume 4102 of *LNCS*, pages 129–144. Springer, 2006. ISBN 978-3-540-38901-9. (Cited on page 205.)

[19] W.M.P. van der Aalst, H.A. Reijers, and M. Song. Discovering social networks from event logs. *Computer Supported Cooperative Work*, 14(6):549–593, 2005. ISSN 0925-9724. (Cited on page 85.)

[20] W.M.P. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142, 2004. (Cited on pages xix, 99, and 147.)

[21] A. Adriansyah. *Aligning Observed and Modeled Behavior*. Ph.D. thesis, Eindhoven University of Technology, 2014. (Cited on pages 98, 109, 111, 113, 122, 123, 189, 197, 199, 221, and 302.)

[22] A. Adriansyah and J.C.A.M. Buijs. Mining process performance from event logs. In M.L. Rosa and P. Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 217–218. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-36284-2. (Cited on pages 109, 122, and 348.)

[23] A. Adriansyah and J.C.A.M. Buijs. Mining process performance from event logs: the BPI Challenge 2012 case study. Technical report, BPM Center Report, No. BPM-12-15, 2012. (Cited on page 350.)

[24] A. Adriansyah, B.F. van Dongen, and W.M.P. van der Aalst. Conformance checking using cost-based fitness analysis. In *EDOC*, pages 55–64. IEEE Computer Society, 2011. ISBN 978-1-4577-0362-1. (Cited on pages 109, 189, and 221.)

[25] A. Adriansyah, N. Sidorova, and B.F. van Dongen. Cost-based fitness in conformance checking. In B. Caillaud, J. Carmona, and K. Hiraishi, editors, *ACSD*, pages 57–66. IEEE, 2011. ISBN 978-0-7695-4387-1. (Cited on pages 189 and 221.)

[26] E. Alpaydin. *Introduction to Machine Learning*. The MIT Press, 2010. (Cited on pages 40 and 49.)

[27] A.K. Alves de Medeiros. *Genetic Process Mining*. Ph.D. thesis, Eindhoven University of Technology, 2006. (Cited on pages 49, 98, 103, 115, 119, and 123.)

[28] A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Using genetic algorithms to mine process models: Representation, operators and results. BETA Working Paper Series, WP 124, Eindhoven University of Technology, Eindhoven, 2004. (Cited on page 40.)

[29] A.K. Alves de Medeiros, A.J.M.M. Weijters, and W.M.P. van der Aalst. Genetic process mining: An experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304, 2007. (Cited on pages xix and 148.)

[30] N. Assy, W. Gaaloul, and B. Defude. Mining configurable process fragments for business process design. In M. Tremblay, D. VanderMeer, M. Rothenberger, A. Gupta, and V. Yoon, editors, *DESRIST*, volume 8463 of *Lecture Notes in Computer Science*, pages 209–224. Springer International Publishing, 2014. ISBN 978-3-319-06700-1. (Cited on page 257.)

[31] E. Badouel and P. Darondeau. Theory of regions. In W. Reisig and G. Rozenberg, editors, *Petri Nets*, volume 1491, pages 529–586. 1996. ISBN 3-540-65306-6. (Cited on page 157.)

[32] J.C.M. Baeten and W.P. Weijland. *Process Algebra*, volume 18 of *Cambridge tracts in theoretical computer science*. Cambridge University Press, Cambridge, 1990. (Cited on page 46.)

[33] C. Baier and J. Katoen. *Principles of Model Checking*. MIT Press, 2007. (Cited on pages 27 and 49.)

[34] J.E. Baker. Reducing bias and inefficiency in the selection algorithm. In *Proceedings of the second international conference on genetic algorithms*, pages 14–21. 1987. (Cited on pages 92 and 139.)

[35] W. Banzhaf, F.D. Francone, R.E. Keller, and P. Nordin. *Genetic Programming: An Introduction: on the Automatic Evolution of Computer Programs and Its Applications*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1998. ISBN 1-55860-510-X. (Cited on pages 86, 87, 88, 137, 138, and 139.)

[36] R.C. Barros, M.P. Basgalupp, A.C.P.L.F. de Carvalho, and A.A. Freitas. A survey of evolutionary algorithms for decision-tree induction. *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, 42(3):291–312, 2012. ISSN 1094-6977. (Cited on page 83.)

[37] R. Bergenthum, J. Desel, R. Lorenz, and S. Mauser. Process mining based on regions of languages. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM*, volume 4714, pages 375–383. 2007. ISBN 978-3-540-75182-3. (Cited on pages 153 and 155.)

[38] S. Betz, D. Eichhorn, S. Hickl, S. Klink, A. Koschmider, Y. Li, A. Oberweis, and R. Trunko. 3d representation of business process models. In P. Loos, M. Nüttgens, K. Turowski, and D. Werth, editors, *MobIS*, volume 141 of *LNI*, pages 73–87. GI, 2008. (Cited on page 275.)

[39] C.G.E. Boender and A.H.G.R. Kan. A bayesian analysis of the number of cells of a multinomial distribution. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 32(1/2):pp. 240–248, 1983. ISSN 00390526. (Cited on page 118.)

[40] R.P.J.C. Bose. *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics*. Ph.D. thesis, Eindhoven University of Technology, 2012. (Cited on pages 81 and 82.)

[41] R.P.J.C. Bose and W.M.P. van der Aalst. Process diagnostics using trace alignment: Opportunities, issues, and challenges. *Information Systems*, 37(2):117–141, 2012. (Cited on page 17.)

[42] R.P.J.C. Bose, W.M.P. van der Aalst, I. Zliobaite, and M. Pechenizkiy. Handling concept drift in process mining. In H. Mouratidis and C. Rolland, editors, *CAiSE*, volume 6741 of *Lecture Notes in Computer Science*, pages 391–405. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-21639-8. (Cited on pages 81 and 82.)

[43] J. Branke, K. Deb, K. Miettinen, and R. Slowinski, editors. *Multiobjective Optimization, Interactive and Evolutionary Approaches [outcome of Dagstuhl seminars].*, volume 5252 of *Lecture Notes in Computer Science*. Springer, 2008. ISBN 978-3-540-88907-6. (Cited on page 90.)

[44] J.C.A.M. Buijs. *Mapping Data Sources to XES in a Generic Way*. Master's thesis, Eindhoven University of Technology, 2010. (Cited on page 350.)

[45] J.C.A.M. Buijs. Environmental permit application process ('WABO'), CoSeLoG project. `http://dx.doi.org/10.4121/uuid:26aba40d-8b2d-435b-b5af-6d4bfbd7a270`, 6 2014. (Cited on page 164.)

[46] J.C.A.M. Buijs. Receipt phase of an environmental permit application process ('WABO'), CoSeLoG project. `http://dx.doi.org/10.4121/uuid:a07386a5-7be3-4367-9535-70bc9e77dbe6`, 6 2014. (Cited on page 164.)

[47] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Towards cross-organizational process mining in collections of process models and their executions. In F. Daniel, K. Barkaoui, S. Dustdar, W.M.P. van der Aalst, J. Mylopoulos, M. Rosemann, M. Shaw, and C. Szyperski, editors, *Business Process Management Workshops (2)*, volume 100 of *Lecture Notes in Business Information Processing*, pages 2–13. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-28115-0. (Cited on pages 20, 259, and 349.)

[48] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. A genetic algorithm for discovering process trees. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012. ISBN 978-1-4673-1510-4. (Cited on page 349.)

[49] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In R. Meersman, H. Panetto, T.S. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I.F. Cruz, editors, *OTM Conferences (1)*, volume 7565 of *Lecture Notes in Computer Science*, pages 305–322. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33605-8. (Cited on pages 142, 145, and 348.)

[50] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Discovering and navigating a collection of process models using multiple quality dimensions. In N. Lohmann, M. Song, and P. Wohed, editors, *Business Process Management Workshops*, volume 171 of *Lecture Notes in Business Information Processing*, pages 3–14. Springer, 2013. ISBN 978-3-319-06256-3. (Cited on page 348.)

[51] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Mining configurable process models from collections of event logs. In F. Daniel, J. Wang, and B. Weber, editors, *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40175-6. (Cited on pages 223 and 349.)

[52] J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, 2014. (Cited on pages 145 and 348.)

[53] J.C.A.M. Buijs, M. La Rosa, H.A. Reijers, B.F. van Dongen, and W.M.P. van der Aalst. Improving business process models using observed behavior. In P. Cudre-Mauroux, P. Ceravolo, and D. Gašević, editors, *SIMPDA*, volume 162 of *Lecture Notes in Business Information Processing*, pages 44–59. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-40918-9. (Cited on pages 203 and 348.)

[54] J.C.A.M. Buijs and H.A. Reijers. Comparing business process variants using models and event logs. In I. Bider, K. Gaaloul, J. Krogstie, S. Nurcan, H.A. Proper, R. Schmidt, and P. Soffer, editors, *BMMDS/EMMSAD*, volume 175 of *Lecture Notes in Business Information Processing*, pages 154–168. Springer, 2014. ISBN 978-3-662-43744-5. (Cited on pages 259 and 348.)

[55] A. Burattin, A. Sperduti, and W.M.P. van der Aalst. Heuristics miners for streaming event data. *CoRR*, abs/1212.6383, 2012. (Cited on page 83.)

[56] M.P. Cabasino, A. Giua, and C. Seatzu. Identification of petri nets from knowledge of their language. *Discrete Event Dynamic Systems*, 17(4):447–474, 2007. (Cited on page 157.)

[57] D.R. Christiansen, M. Carbone, and T.T. Hildebrandt. Formal semantics and implementation of BPMN 2.0 inclusive gateways. In M. Bravetti and T. Bultan, editors, *WS-FM*, volume 6551 of *Lecture Notes in Computer Science*, pages 146–160. Springer Berlin Heidelberg, 2010. ISBN 978-3-642-19588-4. (Cited on page 39.)

[58] J.E. Cook and A.L. Wolf. Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176, 1999. (Cited on pages 98, 113, and 123.)

[59] CoSeLoG. Configurable services for local governments (coselog) project home page (last accessed 2014-04-24). www.win.tue.nl/coselog. (Cited on page 20.)

[60] K. Deb. Multi-objective optimization. In E. Burke and G. Kendall, editors, *Search Methodologies*, pages 273–316. Springer US, 2005. ISBN 978-0-387-28356-2. (Cited on pages 89 and 90.)

[61] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan. A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. *Lecture notes in computer science*, 1917:849–858, 2000. (Cited on page 93.)

[62] J. Dehnert and P. Rittgen. Relaxed soundness of business processes. In K. Dittrich, A. Geppert, and M. Norrie, editors, *CAiSE*, volume 2068, pages 157–170. 2001. ISBN 3-540-42215-3. (Cited on page 37.)

[63] R.M. Dijkman, M. Dumas, B.F. van Dongen, R. Käärik, and J. Mendling. Similarity of business process models: Metrics and evaluation. *Information Systems*, 36(2):498 – 516, 2011. ISSN 0306-4379. (Cited on pages 205 and 206.)

[64] R.M. Dijkman, M. Dumas, and C. Ouyang. Semantics and analysis of business process models in bpmn. *Information & Software Technology*, 50(12):1281–1294, 2008. ISSN 0950-5849. (Cited on pages 39 and 50.)

[65] B.F. van Dongen and W.M.P. van der Aalst. A meta model for process mining data. In J. Casto and E. Teniente, editors, *EMOI-INTEROP*, volume 2 of *CEUR Workshop Proceedings*, pages 309–320. FEUP, Porto, Portugal, 2005. (Cited on page 34.)

[66] B.F. van Dongen and W.M.P. van der Aalst. Multi-phase mining: Aggregating instances graphs into EPCs and Petri nets. In D. Marinescu, editor, *Proceedings of the Second International Workshop on Applications of Petri Nets to Coordination, Workflow and Business Process Management*, pages 35–58. Florida International University, Miami, Florida, USA, 2005. (Cited on pages xix and 156.)

[67] B.F. van Dongen, W.M.P. van der Aalst, and H.M.W. Verbeek. Verification of EPCs: Using reduction rules and petri nets. In O. Pastor and J.F. e Cunha, editors, *CAiSE*, volume 3520 of *Lecture Notes in Computer Science*, pages 372–386. Springer Berlin Heidelberg, 2005. ISBN 3-540-26095-1. (Cited on page 36.)

[68] B.F. van Dongen, R.M. Dijkman, and J. Mendling. Measuring similarity between business process models. In *Proceedings of CAiSE*, volume 5074

of *LNCS*, pages 450–464. Springer, 2008. (Cited on pages 119, 123, and 205.)

[69] B.F. van Dongen, J. Mendling, and W.M.P. van der Aalst. Structural patterns for soundness of business process models. In *EDOC*, EDOC '06, pages 116–128. IEEE Computer Society, Washington, DC, USA, 2006. ISBN 0-7695-2558-X. (Cited on pages 11, 36, 119, and 123.)

[70] A. Dreiling, M. Rosemann, W.M.P. van der Aalst, L. Heuser, and K. Schulz. Model-based software configuration: patterns and languages. *EJIS*, 15(6):583–600, 2006. (Cited on page 224.)

[71] A. Dreiling, M. Rosemann, W.M.P. van der Aalst, W. Sadiq, and S. Khan. Model-driven process configuration of enterprise systems. In O. Ferstl, E. Sinz, S. Eckert, and T. Isselhorst, editors, *Wirtschaftsinformatik*, pages 687–706. Physica-Verlag HD, 2005. ISBN 978-3-7908-1574-0. (Cited on page 224.)

[72] M. Dumas, W.M.P. van der Aalst, and A.H.M. ter Hofstede. *Process-Aware Information Systems: Bridging People and Software Through Process Technology*. Wiley, 2005. ISBN 978-0-471-66306-5. (Cited on page 1.)

[73] D. Dyer. Watchmaker framework version 0.7.1. http://watchmaker.uncommons.org/, 2010. (Cited on page 288.)

[74] M. van Eck. *Alignment-based Process Model Repair and its Application to the Evolutionary Tree Miner*. Master's thesis, Eindhoven University of Technology, 2013. (Cited on pages 128, 129, 134, 135, 136, and 137.)

[75] M. van Eck, J.C.A.M. Buijs, and B.F. van Dongen. Alignment-based process model repair. In *BPI 2014 Workshop*. 2014. (Cited on pages 128, 129, 134, 135, and 136.)

[76] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures - part 1 and part 2. *Acta Informatica*, 27(4):315–368, 1989. (Cited on page 157.)

[77] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer Verlag, 2003. ISBN 978-3-662-05094-1. (Cited on pages 75, 77, 86, 87, 92, 94, 137, 139, and 197.)

[78] E. Esgin and P. Senkul. Delta analysis: A hybrid quantitative approach for measuring discrepancies between business process models. In E. Corchado, M. Kurzynski, and M. Wozniak, editors, *HAIS (1)*, volume 6678 of

Lecture Notes in Computer Science, pages 296–304. Springer, 2011. ISBN 978-3-642-21218-5. (Cited on page 275.)

[79] D. Fahland and W.M.P. van der Aalst. Repairing process models to reflect reality. In A.P. Barros, A. Gal, and E. Kindler, editors, *BPM*, volume 7481 of *LNCS*, pages 229–245. Springer, 2012. ISBN 978-3-642-32884-8. (Cited on pages 221 and 222.)

[80] D. Fahland and W.M.P. van der Aalst. Model repair - aligning process models to reality. *Information Systems*, (0):–, 2013. ISSN 0306-4379. (Cited on pages 221 and 222.)

[81] R. Fehling. A concept of hierarchical Petri nets with building blocks. In G. Rozenberg, editor, *Applications and Theory of Petri Nets*, volume 674 of *Lecture Notes in Computer Science*, pages 148–168. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-56689-2. (Cited on page 50.)

[82] H.G. Fill. Using semantically annotated models for supporting business process benchmarking. In J. Grabis and M. Kirikova, editors, *BIR*, volume 90 of *Lecture Notes in Business Information Processing*, pages 29–43. Springer, 2011. ISBN 978-3-642-24510-7. (Cited on page 274.)

[83] D.B. Fogel and J. Atmar. Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. *Biological Cybernetics*, 63:111–114, 1990. (Cited on page 137.)

[84] W. Fokkink. Process algebra: An algebraic theory of concurrency. In S. Bozapalidis and G. Rahonis, editors, *CAI*, volume 5725 of *Lecture Notes in Computer Science*, pages 47–77. Springer, 2009. ISBN 978-3-642-03563-0. (Cited on pages 46, 48, and 49.)

[85] M.M. Gaber, A.B. Zaslavsky, and S. Krishnaswamy. Mining data streams: a review. *SIGMOD Record*, 34(2):18–26, June 2005. ISSN 0163-5808. (Cited on page 83.)

[86] M. Gambini, M. La Rosa, S. Migliorini, and A.H.M. ter Hofstede. Automated error correction of business process models. In S. Rinderle-Ma, F. Toumani, and K. Wolf, editors, *BPM*, volume 6896 of *LNCS*, pages 148–165. Springer, 2011. ISBN 978-3-642-23058-5. (Cited on page 221.)

[87] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 edition, November 1994. ISBN 0201633612. (Cited on page 291.)

[88] R. Glabbeek and W. Weijland. Branching time and abstraction in bisimulation semantics. *J. ACM*, 43(3):555–600, 1996. (Cited on page 57.)

[89] S. Goedertier, D. Martens, J. Vanthienen, and B. Baesens. Robust process discovery with artificial negative events. *Journal of Machine Learning Research*, 10:1305–1340, June 2009. ISSN 1532-4435. (Cited on pages 98, 112, 116, 122, and 123.)

[90] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, volume 412. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1st edition, 1989. ISBN 0201157675. (Cited on page 197.)

[91] F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Merging event-driven process chains. In R. Meersman and Z. Tari, editors, *OTM Conferences (1)*, volume 5331 of *Lecture Notes in Computer Science*, pages 418–426. Springer Verlag, Berlin Heidelberg, November 2008. ISBN 978-3-540-88870-3. (Cited on page 257.)

[92] F. Gottschalk, W.M.P. van der Aalst, and M.H. Jansen-Vullers. Mining reference process models and their configurations. In R. Meersman, Z. Tari, and P. Herrero, editors, *OTM Workshops*, volume 5333 of *Lecture Notes in Computer Science*, pages 263–272. Springer Verlag, Berlin Heidelberg, November 2008. ISBN 978-3-540-88874-1. (Cited on pages 230, 232, and 257.)

[93] F. Gottschalk, W.M.P. van der Aalst, M.H. Jansen-Vullers, and M. La Rosa. Configurable workflow models. *International Journal of Cooperative Information Systems*, 17(2):177–221, 2008. (Cited on pages 80, 223, 224, and 227.)

[94] F. Gottschalk and M. La Rosa. Process configuration. In A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell, editors, *Modern Business Process Automation*, pages 459–487. Springer, 2010. ISBN 978-3-642-03120-5. (Cited on page 80.)

[95] Government of the Netherlands. List of dutch municipalities (in Dutch, last accessed 2014-04-24). `http://almanak.overheid.nl/categorie/2/Gemeenten_A-Z/`. (Cited on page 18.)

[96] C.W. Günther. *Process Mining in Flexible Environments*. Ph.D. thesis, Eindhoven University of Technology, September 2009. (Cited on pages 12, 13, 46, 48, 49, and 103.)

[97] C.W. Günther and W.M.P. van der Aalst. Fuzzy mining - adaptive process simplification based on multi-perspective metrics. In G. Alonso, P. Dadam, and M. Rosemann, editors, *BPM*, volume 4714 of *Lecture Notes in Computer Science*, pages 328–343. Springer, 2007. ISBN 978-3-540-75182-3. (Cited on pages 12 and 46.)

[98] A. Hallerbach, T. Bauer, and M. Reichert. Capturing variability in business process models: the Provop approach. *Journal of Software Maintenance*, 22(6-7):519–546, 2010. (Cited on pages 226 and 227.)

[99] A. Hallerbach, T. Bauer, and M. Reichert. Configuration and management of process variants. In *Handbook on Business Process Management 1*. Springer, 2010. (Cited on page 274.)

[100] P.J.B. Hancock. An empirical comparison of selection methods in evolutionary algorithms. In T. Fogarty, editor, *Evolutionary Computing, AISB Workshop*, volume 865 of *Lecture Notes in Computer Science*, pages 80–94. Springer Berlin Heidelberg, 1994. ISBN 978-3-540-58483-4. (Cited on pages 92 and 139.)

[101] A.H.M. ter Hofstede, W.M.P. van der Aalst, M. Adams, and N. Russell. *Modern Business Process Automation - YAWL and its Support Environment*. Springer, 2010. ISBN 978-3-642-03120-5. (Cited on pages 36, 41, and 49.)

[102] G.S. Hornby, A. Globus, D.S. Linden, and J.D. Lohn. Automated antenna design with evolutionary algorithms. 2006. (Cited on page 76.)

[103] K. Jensen and G. Rozenberg. *High-level Petri nets: Theory and Applications*. springer, 1991. (Cited on page 50.)

[104] T. Jin, J. Wang, and L. Wen. Efficient retrieval of similar business process models based on structure - (short paper). In R. Meersman, T.S. Dillon, P. Herrero, A. Kumar, M. Reichert, L. Qing, B.C. Ooi, E. Damiani, D.C. Schmidt, J. White, M. Hauswirth, P. Hitzler, and M.K. Mohania, editors, *OTM Conferences (1)*, volume 7044 of *LNCS*, pages 56–63. Springer, 2011. ISBN 978-3-642-25108-5. (Cited on page 205.)

[105] Y. Jin. A comprehensive survey of fitness approximation in evolutionary computation. *Soft Comput.*, 9(1):3–12, 2005. Fitness metrics requirements: survey of approximation. (Cited on page 88.)

[106] Y.C. Juan and C. Ou-Yang. A process logic comparison approach to support business process benchmarking. *The International Journal of Advanced Manufacturing Technology*, 26(1-2), 2005. (Cited on page 274.)

[107] A.J. Keane and S.M. Brown. The design of a satellite boom with enhanced vibration performance using genetic algorithm techniques. In I.C. Parmee, editor, *Proceedings of the Conference on Adaptive Computation in Engineering Design and Control*, pages 107–113. 1996. (Cited on page 76.)

[108] E. Kindler. On the semantics of EPCs: A framework for resolving the vicious circle. In J. Desel, B. Pernici, and M. Weske, editors, *Business Process Management*, volume 3080 of *Lecture Notes in Computer Science*, pages 82–97. Springer Berlin Heidelberg, 2004. ISBN 978-3-540-22235-4. (Cited on pages 38, 39, 43, and 50.)

[109] N. Kleiner. Delta analysis with workflow logs: aligning business process prescriptions and their reality. *Requirements Engineering*, 10(3):212–222, 2005. (Cited on page 275.)

[110] J.R. Koza. *Genetic programming - on the programming of computers by means of natural selection*. Complex adaptive systems. MIT Press, 1993. ISBN 978-0-262-11170-6. (Cited on page 88.)

[111] S. Kriglstein, G. Wallner, and S. Rinderle-Ma. A visualization approach for difference analysis of process models and instance traffic. In F. Daniel, J. Wang, and B. Weber, editors, *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 219–226. Springer, 2013. ISBN 978-3-642-40175-6. (Cited on page 275.)

[112] V. Kulkarni and S. Barat. Business process families using model-driven techniques. In M. zur Muehlen and J. Su, editors, *Business Process Management Workshops*, volume 66 of *Lecture Notes in Business Information Processing*, pages 314–325. Springer, 2010. ISBN 978-3-642-20510-1. (Cited on page 226.)

[113] A. Kumar and W. Yao. Design and management of flexible process variants using templates and rules. *Computers in Industry*, 63(2):112–130, 2012. (Cited on page 226.)

[114] A. Kumar and J.L. Zhao. Workflow support for electronic commerce applications. *Decision Support Systems*, 32(3):265–278, 2002. (Cited on page 51.)

[115] M. Kunze, M. Weidlich, and M. Weske. Behavioral similarity - a proper metric. In S. Rinderle-Ma, F. Toumani, and K. Wolf, editors, *BPM*, volume 6896 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2011. ISBN 978-3-642-23058-5. (Cited on page 205.)

[116] M. La Rosa, W.M.P. van der Aalst, M. Dumas, and F.P. Milani. Business process variability modeling : A survey, 2013. ACM Computing Surveys. (Cited on pages 223 and 224.)

[117] M. La Rosa, M. Dumas, A.H.M. ter Hofstede, and J. Mendling. Configurable multi-perspective business process models. *Information Systems*, 36(2):313–340, 2011. (Cited on page 224.)

[118] M. La Rosa, M. Dumas, R. Uba, and R.M. Dijkman. Business process model merging: An approach to business process consolidation. *ACM Transactions on Software Engineering and Methodology*, 22(2):11, 2013. (Cited on pages 205 and 257.)

[119] M. La Rosa, A.H.M. ter Hofstede, P. Wohed, H.A. Reijers, J. Mendling, and W.M.P. van der Aalst. Managing process model complexity via concrete syntax modifications. *IEEE Transactions on Industrial Informatics*, 7(2):255–265, 2011. (Cited on page 105.)

[120] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering block-structured process models from event logs - a constructive approach. In J.M. Colom and J. Desel, editors, *Petri Nets*, volume 7927 of *Lecture Notes in Computer Science*, pages 311–329. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-38696-1. (Cited on page 130.)

[121] S.J.J. Leemans, D. Fahland, and W.M.P. van der Aalst. Discovering block-structured process models from incomplete event logs. In *Petri Nets 2014*, page to appear. 2014. (Cited on pages 130 and 152.)

[122] M. de Leoni, J.C.A.M. Buijs, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Facilitating process analysis through visualising process history: Experiences with a dutch municipality. (Cited on page 350.)

[123] B.S. Lerner, L.J. Osterweil, S.M. Sutton Jr., and A.E. Wise. Programming process coordination in little-jil. In V. Gruhn, editor, *EWSPT*, volume 1487 of *Lecture Notes in Computer Science*, pages 127–131. Springer, 1998. ISBN 3-540-64956-5. (Cited on page 51.)

[124] C. Li. *Mining Process Model Variants: Challenges, Techniques, Examples*. Ph.D. thesis, University of Twente, November 2010. (Cited on pages 205, 221, and 257.)

[125] C. Li, M. Reichert, and A. Wombacher. Discovering reference models by mining process variants using a heuristic approach. In U. Dayal, J. Eder, J. Koehler, and H.A. Reijers, editors, *BPM*, volume 5701 of *Lecture Notes in Computer Science*, pages 344–362. Springer, 2009. ISBN 978-3-642-03847-1. (Cited on pages 205, 221, and 257.)

[126] C. Li, M. Reichert, and A. Wombacher. The MinAdept clustering approach for discovering reference process models out of process variants. *International Journal of Cooperative Information Systems*, 19(3-4):159–203, 2010. (Cited on pages 205, 221, and 257.)

[127] R. Lu and S.W. Sadiq. Managing process variants as an information resource. In S. Dustdar, J.L. Fiadeiro, and A.P. Sheth, editors, *Business Process Management*, volume 4102 of *Lecture Notes in Computer Science*, pages 426–431. Springer, 2006. ISBN 3-540-38901-6. (Cited on page 274.)

[128] R. Lu and S.W. Sadiq. On the discovery of preferred work practice through business process variants. In C. Parent, K.D. Schewe, V.C. Storey, and B. Thalheim, editors, *ER*, volume 4801 of *Lecture Notes in Computer Science*, pages 165–180. Springer, 2007. ISBN 978-3-540-75562-3. (Cited on page 274.)

[129] S. Luke. Two fast tree-creation algorithms for genetic programming. *IEEE Transactions on Evolutionary Computation*, 4(3):274–283, September 2000. ISSN 1089-778X. (Cited on page 77.)

[130] J. Mendling. *Metrics for Process Models: Empirical Foundations of Verification, Error Prediction, and Guidelines for Correctness*, volume 6 of *Lecture Notes in Business Information Processing*. Springer, 2008. ISBN 978-3-540-89223-6. (Cited on page 263.)

[131] J. Mendling and M. Nüttgens. EPC markup language (EPML): an XML-based interchange format for event-driven process chains (EPC). *Information Systems and e-Business Management*, 4(3):245–263, 2006. (Cited on page 50.)

[132] 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. ISSN 0950-5849. (Cited on pages 39 and 56.)

[133] J. Mendling, H.M.W. Verbeek, B.F. van Dongen, W.M.P. van der Aalst, and G. Neumann. Detection and prediction of errors in EPCs of the SAP reference model. *Data & Knowledge Engineering*, 64(1):312–329, January 2008. ISSN 0169-023X. (Cited on pages 11, 36, 105, 107, and 123.)

[134] Z. Michalewicz. *Genetic algorithms and data structures - evolution programs (3. ed.)*. Springer, 1996. ISBN 978-3-540-60676-5. (Cited on page 197.)

[135] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Notes in Computer Science*. Springer, 1980. ISBN 3-540-10235-3. (Cited on pages 46 and 48.)

[136] D.L. Moody. The "physics" of notations: Toward a scientific basis for constructing visual notations in software engineering. *IEEE Transactions on Software Engineering*, 35(6), 2009. (Cited on page 275.)

[137] F. Mosteller and J.W. Tukey. Data analysis, including statistics. In *Handbook of Social Psychology, Vol. 2*. 1968. (Cited on page 119.)

[138] J. Munoz-Gama and J. Carmona. Enhancing precision in process conformance: Stability, confidence and severity. In N. Chawla, I. King, and A. Sperduti, editors, *CIDM*, pages 184–191. IEEE, Paris, France, April 2011. ISBN 978-1-4244-9925-0. (Cited on pages 114, 123, and 301.)

[139] T. Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989. (Cited on pages 28 and 49.)

[140] T. Nguyen, A.W. Colman, and J. Han. Modeling and managing variability in process-based service compositions. In G. Kappel, Z. Maamar, and H.R.M. Nezhad, editors, *ICSOC*, volume 7084 of *Lecture Notes in Computer Science*, pages 404–420. Springer, 2011. ISBN 978-3-642-25534-2. (Cited on page 226.)

[141] Y. Oirschot. *Using Trace Clustering for Configurable Process Discovery Explained by Event Log Data*. Master's thesis, Eindhoven University of Technology, August 2014. (Cited on pages 81 and 301.)

[142] OMG. BPMN 2.0 by example. http://www.omg.org/spec/BPMN/2.0/examples/PDF/10-06-02.pdf, 6 2010. (Cited on page 31.)

[143] OMG. Business Process Model and Notation (bpmn) version 2.0. Object Management Group, http://www.omg.org/spec/BPMN/2.0/, formal/2011-01-03, 2011. (Cited on pages 12, 31, 36, and 49.)

[144] C. Ouyang, W.M.P. van der Aalst, M. Dumas, A.H. ter Hofstede, and M.L. Rosa. Service-oriented processes : an introduction to BPEL. In J. Cardoso, editor, *Semantic Web services : theory, tools, and applications*, pages 155–188. Information Science Reference (IGI Global), Hershey, PA, 2007. For more information about this book please refer to the publisher's website (see link) or contact the author. (Cited on page 51.)

[145] C. Ouyang, M. Dumas, W.M.P. van der Aalst, A.H.M. ter Hofstede, and J. Mendling. From business process models to process-oriented software systems. *ACM Transactions on Software Engineering and Methodology*, 19(1):2:1–2:37, August 2009. ISSN 1049-331X. (Cited on page 68.)

[146] V. Pareto. *Cours D'Economie Politique, volume I and II*. F. Rouge, Lausanne, 1896. (Cited on page 90.)

[147] M. Pawlik and N. Augsten. RTED: A robust algorithm for the tree edit distance. *CoRR*, abs/1201.0230, 2012. (Cited on pages 206 and 207.)

[148] A. Polyvyanyy, L. García-Bañuelos, and M. Dumas. Structuring acyclic process models. *Information Systems*, 37(6):518 – 538, 2012. ISSN 0306-4379. BPM 2010. (Cited on page 68.)

[149] A. Polyvyanyy, L. García-Bañuelos, D. Fahland, and M. Weske. Maximal structuring of acyclic process models. *Arxiv preprint arXiv:1108.2384*, 2011. (Cited on page 68.)

[150] H.A. Reijers. *Design and Control of Workflow Processes: Business Process Management for the Service Industry*, volume 2617 of *Lecture Notes in Computer Science*. Springer, 2003. ISBN 3-540-01186-2. (Cited on page 85.)

[151] H.A. Reijers, T. Freytag, J. Mendling, and A. Eckleder. Syntax highlighting in business process models. *Decision Support Systems*, 51(3):339–349, 2011. (Cited on page 275.)

[152] H.A. Reijers and J. Mendling. A study into the factors that influence the understandability of business process models. *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, 41(3):449–462, 2011. (Cited on page 105.)

[153] A. Rolstadås. *Performance Management: A Business Process Benchmarking Approach*. Chapman & Hall, 1995. (Cited on page 274.)

[154] M. Rosemann and W.M.P. van der Aalst. A configurable reference modeling language. *Information Systems*, 32(1):1–23, 2007. (Cited on pages 80, 223, 224, and 225.)

[155] A. Rozinat. *Process Mining: Conformance and Extension*. Ph.D. thesis, Eindhoven University of Technology, November 2010. (Cited on pages 88, 98, 111, 112, 122, and 123.)

[156] A. Rozinat and W.M.P. van der Aalst. Decision mining in ProM. In S. Dustdar, J. Fiadeiro, and A. Sheth, editors, *Business Process Management*, volume 4102, pages 420–425. 2006. ISBN 3-540-38901-6. (Cited on page 83.)

[157] 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. (Cited on pages 98, 112, 115, 119, 122, and 123.)

[158] A. Rozinat, R.S. Mans, M. Song, and W.M.P. van der Aalst. Discovering simulation models. *Information Systems*, 34(3):305–327, 2009. (Cited on page 84.)

[159] A.W. Scheer. *Business Process Engineering, Reference Models for Industrial Enterprises*. Springer-Verlag, Berlin, 1994. (Cited on pages 12, 43, and 49.)

[160] A. Schnieders and F. Puhlmann. Variability mechanisms in e-business process families. In W. Abramowicz and H.C. Mayr, editors, *BIS*, volume 85 of *LNI*, pages 583–601. GI, 2006. ISBN 3-88579-179-X. (Cited on page 226.)

[161] D.M.M. Schunselaar, H.M.W. Verbeek, W.M.P. van der Aalst, and H.A. Reijers. Creating sound and reversible configurable process models using CoSeNets. In W. Abramowicz, D. Kriksciuniene, and V. Sakalauskas,

editors, *BIS*, volume 117 of *Lecture Notes in Business Information Processing*, pages 24–35. Springer, 2012. ISBN 978-3-642-30358-6. (Cited on pages xviii, 56, 72, 232, and 257.)

[162] D.M.M. Schunselaar, H.M.W. Verbeek, H.A. Reijers, and W.M.P. van der Aalst. YAWL in the cloud: Supporting process sharing and variability. In *BPMC 2014 Workshop*. 2014. (Cited on page 305.)

[163] N.I. Seneratna. Genetic algorithms: The crossover-mutation debate (bachelor literature survey), 2005. (Cited on pages 137 and 138.)

[164] M. Solé and J. Carmona. Process mining from a basis of state regions. In J. Lilius and W. Penczek, editors, *Petri Nets*, volume 6128, pages 226–245. 2010. ISBN 978-3-642-13674-0. (Cited on page 157.)

[165] M. Song and W.M.P. van der Aalst. Supporting process mining by showing events at a glance. In K. Chari and A. Kumar, editors, *Proceedings of 17th Annual Workshop on Information Technologies and Systems (WITS 2007)*, pages 139–145. Montreal, Canada, December 2007. (Cited on pages 17 and 268.)

[166] M. Song and W.M.P. van der Aalst. Towards comprehensive support for organizational mining. *Decision Support Systems*, 46(1):300–317, 2008. (Cited on pages 17 and 268.)

[167] W.M. Spears and V. Anand. A study of crossover operators in genetic programming. In Z. Ras and M. Zemankova, editors, *ISMIS*, volume 542 of *Lecture Notes in Computer Science*, pages 409–418. Springer Berlin Heidelberg, 1991. ISBN 978-3-540-54563-7. (Cited on page 138.)

[168] F. Teuteberg, M. Kluth, F. Ahlemann, and S. Smolnik. Semantic process benchmarking to improve process performance. *Benchmarking: An International Journal*, 20(4), 2013. (Cited on page 274.)

[169] S. Thatte. Xlang: Web services for business process design. *Microsoft Corporation*, 2001, 2001. (Cited on page 51.)

[170] J. Vanhatalo, H. Völzer, and J. Koehler. The refined process structure tree. *Data and Knowledge Engineering*, 68(9):793 – 818, 2009. ISSN 0169-023X. Sixth International Conference on Business Process Management (BPM 2008) Five selected and extended papers. (Cited on page 68.)

[171] D.A. van Veldhuizen and G.B. Lamont. Evolutionary computation and convergence to a pareto front. In *Late Breaking Papers at the Genetic Programming 1998 Conference*, pages 221–228. 1998. (Cited on page 90.)

[172] H.M.W. Verbeek, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Verifying workflows with cancellation regions and OR-joins: An approach based on relaxed soundness and invariants. *The Computer Journal*, 50(3):294–314, 2007. (Cited on page 36.)

[173] H.M.W. Verbeek, T. Basten, and W.M.P. van der Aalst. Diagnosing workflow processes using Woflan. *The Computer Journal*, 44(4):246–279, 2001. (Cited on pages 36 and 37.)

[174] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. ProM 6: The process mining toolkit. In *Proc. of BPM Demonstration Track 2010*, volume 615, pages 34–39. CEUR-WS.org, 2010. (Cited on pages 277 and 350.)

[175] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES tools, 2010. (Cited on page 350.)

[176] H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In P. Soffer and E. Proper, editors, *CAiSE Forum*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer, 2010. ISBN 978-3-642-17721-7. (Cited on pages 34, 277, 285, 290, 293, and 350.)

[177] J. de Weerdt, M. de Backer, J. Vanthienen, and B. Baesens. A robust f-measure for evaluating discovered process models. In *CIDM*, pages 148–155. IEEE, 2011. ISBN 978-1-4244-9925-0. (Cited on pages 112 and 122.)

[178] M. Weidlich, R.M. Dijkman, and J. Mendling. The ICoP framework: Identification of Correspondences between Process models. In B. Pernici, editor, *CAiSE*, volume 6051 of *Lecture Notes in Computer Science*, pages 483–498. Springer, Springer, 2010. ISBN 978-3-642-13093-9. (Cited on page 275.)

[179] M. Weidlich, A. Polyvyanyy, N. Desai, and J. Mendling. Process compliance measurement based on behavioural profiles. In B. Pernici, editor, *CAiSE*, volume 6051 of *Lecture Notes in Computer Science*, pages 499–514. Springer, Springer, 2010. ISBN 978-3-642-13093-9. (Cited on page 99.)

[180] A.J.M.M. Weijters, W.M.P. van der Aalst, and A.K.A. de Medeiros. Process mining with the heuristics miner-algorithm. *Technische Universiteit Eindhoven, Tech. Rep. WP*, 166, 2006. (Cited on pages xix, 12, 40, 45, 103, 149, and 150.)

[181] A.J.M.M. Weijters and J.T.S. Ribeiro. Flexible heuristics miner (FHM). In *CIDM*, pages 310–317. IEEE, 2011. ISBN 978-1-4244-9925-0. (Cited on pages 12, 45, and 49.)

[182] J.M.E.M. van der Werf, B.F. van Dongen, C.A.J. Hurkens, and A. Serebrenik. Process discovery using integer linear programming. *Fundamenta Informaticae*, 94:387–412, 2010. (Cited on pages xix and 151.)

[183] D. Wodtke and G. Weikum. A formal foundation for distributed workflow execution based on state charts. In F. Afrati and P. Kolaitis, editors, *ICDT*, volume 1186 of *Lecture Notes in Computer Science*, pages 230–246. Springer Berlin Heidelberg, 1997. ISBN 978-3-540-62222-2. (Cited on page 39.)

[184] R. Wolffensperger. *Static and dynamic visualization of quality and performance dimensions on Process Trees*. Master's thesis, Eindhoven University of Technology, 2014. (Cited on page 302.)

[185] M.T. Wynn, D. Edmond, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Achieving a general, formal and decidable approach to the OR-join in workflow using reset nets. In G. Ciardo and P. Darondeau, editors, *ICATPN*, volume 3536 of *Lecture Notes in Computer Science*, pages 423–443. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-26301-2. (Cited on pages 38, 39, and 50.)

[186] A. Yakovlev, M. Kishinevsky, A. Kondratyev, L. Lavagno, and M. Pietkiewicz-Koutny. On the models for asynchronous circuit behaviour with OR causality. *Formal Methods in System Design*, 9(3):189–233, 1996. ISSN 0925-9856. (Cited on pages 38 and 50.)

[187] H. Zha, J. Wang, L. Wen, C. Wang, and J. Sun. A workflow net similarity measure based on transition adjacency relations. *Computers in Industry*, 61(5):463–471, 2010. (Cited on page 205.)

[188] E. Zitzler, M. Laumanns, and L. Thiele. SPEA2: Improving the strength pareto evolutionary algorithm for multiobjective optimization. In K. Giannakoglou, D. Tsahalis, J. Periaux, K. Papaliliou, and T. Fogarty, editors,

Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems. Proceedings of the EUROGEN2001 Conference, Athens, Greece, September 19-21, 2001, pages 95–100. International Center for Numerical Methods in Engineering (CIMNE), Barcelona, Spain, 2002. (Cited on page 93.)

[189] M. Zur Muehlen and J. Recker. How much language is enough? Theoretical and practical use of the business process modeling notation. In Z. Bellahsène and M. Léonard, editors, *Advanced Information Systems Engineering*, volume 5074 of *Lecture Notes in Computer Science*, pages 465–479. Springer Berlin Heidelberg, 2008. ISBN 978-3-540-69533-2. (Cited on page 32.)

Index

- τ, 27
- α-algorithm, 147
- $\wedge$-operator, 51
  - language, 53
- $\circlearrowleft$-operator, 51
  - language, 53
- $\vee$-operator, 51
  - language, 53
- $\rightarrow$-operator, 51
  - language, 53
- $\leftarrow$-operator, 51
  - language, 53
- $\times$-operator, 51
  - language, 53

- Activity, 27
- Activity universe, 27
- Alignment, 108
- Alignment matrix, 259, 265
  - visualization, 265
- Anytime behavior, 86

- Bags, *see* Multi-sets
- Bloat, 87
- Block-structured process models, 50
- BPMN, 31
  - gateways, 31
- Business Process Modeling and Notation, *see* BPMN

- C-EPC, 224
- C-net, *see* Causal net
- C-YAWL, 224
- Calculus of Communicating Systems, *see* CCS
- Candidate evaluation, 77, 89
- Candidate selection, 91, 138
  - fitness proportional selection, 92
  - ranking selection, 92
  - roulette wheel selection, 92
  - sigma scaling selection, 92
  - stochastic universal sampling, 92
  - tournament selection, 92
- Case, 2
- Causal net, 43
- CCS, 46, 48
- Change operators, 77
- Comparison framework, 259, 261
  - event log metric, 261, 263
  - comparison metric, 261, 263

process model metric, 261, 263
Comparison table, 260, *see* Comparison framework
Completeness, 103
Concept drift, 81
gradual drift, 82
incremental drift, 82
recurring drift, 82
sudden drift, 81
Configurable EPC, *see* C-EPC
Configurable process model, 16, 224
Configurable Services for Local Governments, *see* CoSeLoG
Configurable YAWL, *see* C-YAWL
CoSeLoG, 19
CoSeNet, 56, 257
Crossover, 77, 94, 137

Edit operation, 205
Elite, 77
EPC, 43
Escaping edges, 114
ETM framework
basic flow, 76
CLI, 285
extension, 288
GUI, 278
implementation, 277
plug-ins, 278
requirements, 85
ETM*c* algorithm, 80, 232
mutation, 233
quality, 233
ETM*context* algorithm, 81
ETM*d* algorithm, 78, 125, 163
ETM*dec* algorithm, 83
ETM*drift* algorithm, 82
ETM*r* algorithm, 79, 203
Event, 2
Event log, 32, 34
completeness, *see* Completeness
definition, 34
Event-Driven Process Chains, *see* EPC
Evolutionary Tree Miner, *see* ETM framework
Exceptional behavior, 103
Exploitation, 86, 140
Exploration, 86, 140
eXtensible Event Stream, *see* XES

Finite-state automaton, 28
Finite-state machine, 28
Frequency of use, 117
FSM, *see* Finite-State-Machine
Function, 26
Fuzzy miner, 46
Fuzzy model, 46

Generalization, 5, 98, 103, 104, 116
frequency of use, *see* Frequency of use
Genetic miner, 148

Heuristics miner, 45, 149
Heuristics net, 45
Hidden Markov Model, 40
HMM, *see* Hidden Markov Model

ILP miner, 151
Inductive Miner, 130, 152
Initial population, 76, 125
advanced trace-model creation, 127
creation, 125
random tree creation, 127
Intron, 87
IS-SaaS, 20

Labeled directed graph, 27
Labeled Petri net, *see* Petri net

Labeled transition system, *see* Transition system
Language-based region theory, 153
Local optimum, 86
Long-term dependency operator, *see* LTD-operator
LTD-operator, 303

Move on log only, 109
Move on model only, 109
Multi-phase miner, 156
Multi-sets, 26
Mutation, 77, 94
  guided, 134
    add behavior, 135
    change behavior, 136
    remove behavior, 134
  random, 131
    change, 133
    node addition, 132
    node removal, 132
    normalization, 133
    remove useless nodes, 133
    shuffle, 133
    tree replacement, 133
MXML, 34

Noise, *see* Exceptional behavior, 103
Non-exclusive choice, 38

Occam‘s Razor, 105
OR, *see* Non-exclusive choice

Pareto front, 90, 145
Petri net, 28
  completed trace, 30
  definition, 29
  labeled, 30
  language, 30
  marking, 28, 29
    final, 29
    initial, 29
    reachable, 30
  place, 28
  token, 28
  transition, 28
    enabled, 30
    fire, 30
Precision, 5, 98, 102, 103, 113
  advanced behavioral appropriateness, 115
  escaping edges, *see* Escaping edges
Premature convergence, 86
Process algebra, 46
Process instance, 32
Process mining, 2
  conformance, 2
  discovery, 2
  extension, 2
  quality, 5
Process model, 27
  block-structured, 50
  error free, *see* Soundness
  expressiveness, 37
    concurrency, 38
    duplicate activities, 38
    non-exclusive choice, *see* Non-exclusive choice
    non-free choice, 38
    silent actions, 38
  quality, 98
  repair, 204
  visualization, 11
Process Tree, 51
  definition, 51
  example, 51
  language, 53
  leaf node, 51

operator node, 51
root, 51
translation, 57
to BMPN, 62
to CCS, 66
to CoSeNet, 66
to EPC, 64
to Petri Net, 57
to YAWL, 60
PROcess Variants by OPtions approach, *see* PROVOP
Projection, 26
ProM, 5
PROVOP, 226

Relaxed soundness, 37
Replay fitness, 5, 98, 102, 103, 108
alignment, *see* Alignment
artificial negative events, 112
streams, 113
token-based, 112
Representational bias, 11
Robust Tree Edit Distance algorithm, *see* RTED algorithm
RTED algorithm, 207
Running examples, 5
exceptional behavior, 6
four variants, 7
non-exceptional behavior, 6

SaaS, *see* Software as a Service
Sequence, 26
concatenation, 26
Set, 25
difference, 26
empty, 25
intersection, 26
non-empty, 25
union, 25
Similarity, 205
behavioral, 205
structural, 205
Simplicity, 5, 98, 103, 105
activity occurrence, 107
ratio of useless nodes, 105
size, 107
Simulation, 84
Software as a Service, 20
Soundness, 11, 36
no dead transitions, 36
option to complete, 36
proper completion, 36
State-based region theory, 157
Stream mining, 83
Synchronous move, 109

Termination, 77, 94, 139
elapsed time, 95
number of generation threshold, 95
quality stagnation, 95
quality threshold, 94
user cancelation, 95
Trace, 32, 34
Transition system, 27
definition, 27
final states, 27
initial states, 27
state, 27
transition, 27

Useless node, 106

XES, 34

YAWL, 41
Yet Another Workflow Language, *see* YAWL

Summary

Flexible Evolutionary Algorithms for Mining Structured Process Models

The goal of process mining is to automatically produce process models that accurately describe processes by considering only an organization's records of its operational processes. Such records are typically captured in the form of event logs, consisting of cases and events related to these cases. Using these event logs process models can be discovered. Over the last decade, many such process discovery techniques have been developed, producing process models in various forms, such as Petri nets, BPMN-models, EPCs, YAWL-models etc. Furthermore, many authors have compared these techniques by focusing on the properties of the models produced, while at the same time the applicability of various techniques have been compared in case-studies. In this thesis we present a new process discovery algorithm: the Evolutionary Tree Miner, or ETM for short. The Evolutionary Tree Miner however has some unique characteristics and features, that are not found in existing process discovery algorithms.

The main property of the Evolutionary Tree Miner is that it always produces a sound (i.e., syntactically correct) process model. Although this is a prerequisite for the process model to be used for further analysis, very few of the existing process discovery algorithms can guarantee this.

Another main feature of the Evolutionary Tree Miner is that it is a flexible algorithm. The four well known quality dimensions in process discovery (replay fitness, precision, generalization and simplicity) are explicitly incorporated in the Evolutionary Tree Miner. Additional quality metrics can be easily added to the Evolutionary Tree Miner. The Evolutionary Tree Miner is able to balance the different provided quality metrics and is able to produce process models that have a specific balance of these quality dimensions, as specified by the user.

The third main feature of the Evolutionary Tree Miner is that it is easily extensible. In this thesis we discuss several scenarios where the Evolutionary Tree Miner can be applied. We discuss the following extensions and applications in more detail in this thesis:

1. The discovery of a collection of process models, each having a unique and superior set of characteristics for the provided quality dimensions (more concretely: the process models are Pareto optimal).

2. Discovery of a process model given a (collection of) normative process models. This allows for the repair of a process model, using the observed behavior.

3. Discovery of a configurable process model that describes multiple event logs, for instance from different organizations, using only one process model and a configuration of the model for each event log.

4. A comparison framework is discussed that allows for the comparison of executions of similar processes. This framework is able to replay the behavior of one organization on the (configurable) process model of another, to provide insights into differences and commonalities between the process execution of the different organizations.

The Evolutionary Tree Miner is implemented as a plug-in for the process mining toolkit ProM. Furthermore, the usage of the different plug-ins created in this thesis is discussed. Additionally, it is shown how the Evolutionary Tree Miner can be extended to include custom quality dimensions.

The Evolutionary Tree Miner, and all of its extensions, are evaluated using both artificial and real-life data sets.

Samenvatting

Flexible Evolutionary Algorithms for Mining Structured Process Models

Het doel van process mining is het automatisch ontdekken van procesmodellen die accuraat processen beschrijven door alleen gebruik te maken van de data van de al uitgevoerde processen. Deze data worden meestal opgeslagen in gebeurtenissenlogboeken, die zaken en gebeurtenissen gerelateerd aan deze zaken opslaan. Op basis van deze gebeurtenissenlogboeken kunnen procesmodellen worden ontdekt. In het laatste decennium zijn veel van deze technieken ontwikkeld die procesmodellen in verschillende vormen produceren zoals Petri nets, BPMN-modellen, EPCs, YAWL-modellen. Veel auteurs hebben deze technieken bovendien vergeleken door te focussen op de eigenschappen van de geproduceerde modellen, terwijl tegelijk de toepasbaarheid van deze technieken is vergeleken in verschillende case studies. In dit proefschrift presenteren we een nieuw algoritme: de Evolutionary Tree Miner (Evolutionaire Boom Ontdekker), of ETM in het kort. Het ETM heeft enkele unieke karakteristieken en mogelijkheden die niet voorkomen in bestaande proces ontdek technieken.

De belangrijkste eigenschap van het ETM is dat het altijd (syntactisch) correcte procesmodellen ontdekt. Ondanks dat dit een vereiste is om het procesmodel te kunnen gebruiken voor verdere analyse, garanderen zeer weinig technieken dit.

Een andere belangrijke eigenschap van het ETM is dat het een flexibel algoritme is. De vier bekende kwaliteitsdimensies in het ontdekken van procesmodellen (naspeelbaarheid, precisie, generalizatie en eenvoud) zijn expliciet ingebouwd in het ETM. Aanvullende kwaliteitsdimensies kunnen gemakkelijk worden toegevoegd aan het ETM. Het ETM kan de verschillende kwaliteitsdimensies balanceren en kan procesmodellen produceren die de kwaliteitsdimensies op een specifieke manier balanceren, zoals gespecificeerd door de gebruiker.

De derde belangrijke eigenschap van het ETM is dat het gemakkelijk uitbreidbaar is. In dit proefschrift bespreken we verschillende scenarios waar het ETM toegepast kan worden. De volgende uitbreidingen en toepassingen worden in dit proefschrift in meer detail besproken:

1. Het ontdekken van een collectie van procesmodellen, die elk een unieke en superieure set van karakteristieken voor elke kwaliteitsdimensie hebben (concreter: de procesmodellen zijn Pareto optimaal).

2. Het ontdekken van een procesmodel op basis van één of meerdere normatieve procesmodellen. Dit maakt het mogelijk om de gegeven modellen te repareren gebruikmakend van het geobserveerde gedrag.

3. Het ontdekken van een configureerbaar procesmodel dat meerdere gebeurtenissenlogboeken beschrijft, bijvoorbeeld van verschillende organisaties, in een enkel procesmodel met een configuratie voor elk gebeurtenissenlogboek.

4. Een vergelijkingsraamwerk dat het vergelijken van de uitvoer van verschillende vergelijkbare processen toestaat. Het raamwerk kan het geobserveerde gedrag van een organisatie naspelen op het (configureerbare) procesmodel van een andere organisatie. Hierdoor worden inzichten in de verschillen en overeenkomsten tussen de uitvoer van de processen van de verschillende organisaties verkregen.

Het ETM is geïmplementeerd in de process minig toolkit ProM. Ook is het gebruik besproken van de verschillende plug-ins die zijn gemaakt in het kader van dit proefschrift. Daarnaast is getoond hoe het ETM uitgebreid kan worden met aangepaste kwaliteitsdimensies.

Het ETM en alle uitbreidingen zijn geëvalueerd op basis van zowel kunstmatige als echte data sets.

Acknowledgments

This thesis marks the end of a four year journey. And although the journey was not always easy, or fun, I have never regretted that I started it. It is also a journey that I would have never undertaken if it were not for my promotor, professor Wil van der Aalst, who asked me to consider doing a Ph.D. Wil, thank you for taking me on this journey and for your guidance, it has been a true pleasure and I hope that we can continue our collaboration. Someone that was also with me throughout this voyage is my co-promotor, Boudewijn van Dongen. Boudewijn, thank you for correcting me, re-aligning me and challenging me, but most of all for being more than a colleague to me. In the past four years Hajo Reijers played the role of mentor, whom I could consult for more general questions. Hajo, thank you for always making time for me and for our broad and open discussions, and for our travels together.

During my Ph.D. I had the opportunity to visit the group of Arthur ter Hofstede at Queensland University of Technology in Brisbane, Australia, for six weeks. Arthur, thank you for having me in your group, taking me out for lunch and diner and for the random talks we had. In my first year as a Ph.D. student Marlon Dumas, now from the university of Tartu, came to visit Eindhoven and I had the pleasure of sharing an office with him for a few days. Marlon, I appreciate your vision and work and I'm looking forward to the challenges you will present me during my defense. I would also like to thank Barbara Hammer from Bielefeld University for her willingness to join my Ph.D. committee. I am really looking forward to your questions at my defense, which I hope will be from a new perspective. Finally I would like to thank Jack van Wijk for reviewing my thesis and especially my visualizations, I appreciate your time and openness to discussion.

All people mentioned so far are part of my Ph.D. committee and as such have read this thesis thoroughly in order to evaluate its contents. I would like to thank all of them for the detailed feedback they provided that shows that they really read and understood my work, even though some still don't agree with everything I have written.

My office mates Dennis Schunselaar and Elham Ramezani Taghiabadi joined my journey up close. Dennis, thank you for our collaboration within the CoSeLoG project, for your critical feedback and for our many (ir)relevant discussions. Elham, thank you for adding a different cultural view to the office, for your social engagement and for your debating skills (even when you were outnumbered two to one you stood your ground). Both of you have become my best friends and I hope we will occupy the 'cool' office together for quite some time to come.

This journey was made much more practical by collaborating with the partners in the CoSeLoG project. Without them this thesis would have had no real-life data, no real-life challenges, no real-life solutions and no real-life case studies.

Next I would like to thank all colleagues from both sides of the information systems group. Some of my colleagues, present and past, deserve a special mention. Special thanks go to both Ine van der Ligt and Riet van Buul for their many life-saving actions and for being the center of the group. Arya Adriansyah, thank you for our nice collaborations and talks about life and research, we must stay in touch! I would also like to thank Eric Verbeek for his technical support and red M&Ms, Xixi Lu for the company in the evenings and weekends, and Sander Leemans and Ronny Mans for looking at steam trains together. And of course a big thank you to all other current colleagues: Alfredo, Bas, Dirk, Eduardo, Felix, George, Julia Kiseleva, Maikel, Massimiliano, Murat, Mykola, Natalia, Natasha, Paul, Rafal, Richard, Shengnan, and Shiva for all the lunch talks and game evenings. I will not forget my former colleagues either: Arjan Mooi, Carmen Bratosin, Christian Stahl, Delia Arsinte, Fabrizio Maggi, Han van der Aa, Helen Schonenberg, Jan Vogelaar, Jan Martijn van der Werf, JC Bose, Joyce Nakatumba, Maja Pešić, and Michael Westergaard. And then there were of course memorable visitors that we spent time with: Anna Kalenkova, Artem Polyvyanyy, Christian Gierds, Cristina Iovino, Jorge Muñoz-Gamma, and Raffaele Conforti. Additionally, I would like to thank Marcello La Rosa, Artem Polyvyanyy, Raffaele Conforti, and Suriadi Suriadi and all other QUT group members for making my visit to Brisbane a memorable experience. Finally, I would also like to thank the master students that I got to supervise: Danny van Heumen, Guido Swinkels, Tatiana Mamaliga, Maikel van Eck, and Yoran van Oirschot. Thank you for the discussions, new perspectives, young enthusiasm, inspiration and work off the main thesis road. And of course to all others that I have met in Eindhoven or abroad, but that I forgot to mention: you all contributed, big or small, to this work.

Special thanks go to Thijs Nugteren and Eva Ploum for being my master study buddies. You made me go where I did not believe I could during my master study. I hope we keep having semi-regular meetings to share our knowledge, experience and life stories.

However, my family joined and supported my journey the most. Mom, dad, thank you for unconditionally supporting me throughout my studies. I would also like to thank my brothers, Tijn and Dirk, for being who you are, for your willingness to help and for growing up with me. Special thanks go to Dirk for his thorough spelling and grammar check of this full thesis. I would also like to thank my parents-in-law, Marléne and George, and my brother-in-law Melvin, for joining me on this journey, and for adding other journeys in the weekends. But most of all I would like to thank my wife and daughter, who joined me when things were good and bad, day in, day out. Debbie, my dear wife, thank you for taking me on many journeys in the past years, for your understanding of the side-effects of chasing a Ph.D., and for your perseverance during the tough times. I would also like to thank Mila, our daughter, for showing me evolution up close and for all the journeys, big and very small, that we made and will make.

Joos Buijs
Eindhoven, August 2014

Curriculum Vitae

Joseph Cornelis Antonius Maria (Joos) Buijs was born on June 29, 1984 in Breda, The Netherlands. He grew up in the town of Prinsenbeek with his parents and two younger brothers. He attended secondary school at the Prisma-Graaf Engelbrecht college in Breda. After finishing secondary school in 2002, he started his physics studies at Eindhoven University of Technology but soon switched to computer science. In 2004 he started the short programme for informatics studies at Avans university of applied sciences, which he finished successfully in 2007. Next he followed the master Business Information Systems at Eindhoven University of Technology. During his master, he was a student assistant at the Laboratory of Quality Software (LaQuSo), where he conducted several projects related to process mining. In 2010 he graduated on the dissertation “Mapping Data Sources to XES in a Generic Way”.

Joos became a PhD candidate in May 2010 when he started on the CoSeLoG project. The overall goal of the CoSeLoG project is to investigate how processes differ between municipalities by analyzing how they are executed, and how these processes can be supported by a shared business process management system. The work of Joos resulted in a number of publications at international conferences and in an international journal. He defended his doctoral thesis, entitled “Flexible Evolutionary Algorithms for Mining Structured Process Models” and supervised by prof.dr.ir. W.M.P. van der Aalst and dr.ir. B.F. van Dongen, on October 28, 2014.

Since June 2014 Joos is working as a postdoctoral researcher, where he assists prof.dr.ir. W.M.P. van der Aalst, scientific head of the Data Science Center Eindhoven (DSC/e), in research, teaching, and industry collaborations. Joos can be reached at `j.c.a.m.buijs@tue.nl`.

List of Publications

Joos Buijs has the following publications:

Journals

- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Quality dimensions in process discovery: The importance of fitness, precision, generalization and simplicity. *International Journal of Cooperative Information Systems*, 2014

Proceedings and Congres Contributions

- J.C.A.M. Buijs and H.A. Reijers. Comparing business process variants using models and event logs. In I. Bider, K. Gaaloul, J. Krogstie, S. Nurcan, H.A. Proper, R. Schmidt, and P. Soffer, editors, *BMMDS/EMMSAD*, volume 175 of *Lecture Notes in Business Information Processing*, pages 154–168. Springer, 2014. ISBN 978-3-662-43744-5

- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Discovering and navigating a collection of process models using multiple quality dimensions. In N. Lohmann, M. Song, and P. Wohed, editors, *Business Process Management Workshops*, volume 171 of *Lecture Notes in Business Information Processing*, pages 3–14. Springer, 2013. ISBN 978-3-319-06256-3

- A. Adriansyah and J.C.A.M. Buijs. Mining process performance from event logs. In M.L. Rosa and P. Soffer, editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 217–218. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-36284-2

- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. On the role of fitness, precision, generalization and simplicity in process discovery. In R. Meersman, H. Panetto, T.S. Dillon, S. Rinderle-Ma, P. Dadam, X. Zhou, S. Pearson, A. Ferscha, S. Bergamaschi, and I.F. Cruz, editors, *OTM Conferences (1)*, volume 7565 of *Lecture Notes in Computer Science*, pages 305–322. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-33605-8

- J.C.A.M. Buijs, M. La Rosa, H.A. Reijers, B.F. van Dongen, and W.M.P. van der Aalst. Improving business process models using observed behavior. In P. Cudre-Mauroux, P. Ceravolo, and D. Gašević, editors, *SIMPDA*, volume 162 of *Lecture Notes in Business Information Processing*, pages 44–59. Springer Berlin Heidelberg, 2012. ISBN 978-3-642-40918-9

- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Mining configurable process models from collections of event logs. In F. Daniel, J. Wang, and B. Weber, editors, *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 33–48. Springer Berlin Heidelberg, 2013. ISBN 978-3-642-40175-6

- W.M.P. van der Aalst, J.C.A.M. Buijs, and B.F. van Dongen. Towards improving the representational bias of process mining. In K. Aberer, E. Damiani, and T. Dillon, editors, *SIMPDA*, volume 116 of *Lecture Notes in Business Information Processing*, pages 39–54. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-34043-7

- W.M.P. van der Aalst, A. Adriansyah, A.K.A. de Medeiros, F. Arcieri, T. Baier, T. Blickle, R.P.J.C. Bose, P. van den Brand, R. Brandtjen, J.C.A.M. Buijs, A. Burattin, J. Carmona, M. Castellanos, J. Claes, J. Cook, N. Costantini, F. Curbera, E. Damiani, M. de Leoni, P. Delias, B.F. van Dongen, M. Dumas, S. Dustdar, D. Fahland, D.R. Ferreira, W. Gaaloul, F. van Geffen, S. Goel, C.W. Günther, A. Guzzo, P. Harmon, A.H.M. ter Hofstede, J. Hoogland, J.E. Ingvaldsen, K. Kato, R. Kuhn, A. Kumar, M. La Rosa, F.M. Maggi, D. Malerba, R.S. Mans, A. Manuel, M. McCreesh, P. Mello, J. Mendling, M. Montali, H.R.M. Nezhad, M. zur Muehlen, J. Munoz-Gama, L. Pontieri, J. Ribeiro, A. Rozinat, H.S. Pérez, R.S. Pérez, M. Sepúlveda, J. Sinur, P. Soffer, M. Song, A. Sperduti, G. Stilo, C. Stoel, K.D. Swenson, M. Talamo, W. Tan, C. Turner, J. Vanthienen, G. Varvaressos, H.M.W. Verbeek, M. Verdonk, R. Vigo, J. Wang, B. Weber, M. Weidlich, T. Weijters, L. Wen, M. Westergaard, and M.T. Wynn. Process mining manifesto. In F. Daniel, K. Barkaoui, and S. Dustdar, editors, *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-28107-5

- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. A genetic algorithm for discovering process trees. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012. ISBN 978-1-4673-1510-4

- J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. Towards cross-organizational process mining in collections of process models and their executions. In F. Daniel, K. Barkaoui, S. Dustdar, W.M.P. van der Aalst,

J. Mylopoulos, M. Rosemann, M. Shaw, and C. Szyperski, editors, *Business Process Management Workshops (2)*, volume 100 of *Lecture Notes in Business Information Processing*, pages 2–13. Springer Berlin Heidelberg, 2011. ISBN 978-3-642-28115-0

- H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES, XESame, and ProM 6. In P. Soffer and E. Proper, editors, *CAiSE Forum*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer, 2010. ISBN 978-3-642-17721-7

- H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. ProM 6: The process mining toolkit. In *Proc. of BPM Demonstration Track 2010*, volume 615, pages 34–39. CEUR-WS.org, 2010

- H.M.W. Verbeek, J.C.A.M. Buijs, B.F. van Dongen, and W.M.P. van der Aalst. XES tools, 2010

Master Thesis

- J.C.A.M. Buijs. *Mapping Data Sources to XES in a Generic Way*. Master's thesis, Eindhoven University of Technology, 2010

Technical Reports (Non-Refereed)

- A. Adriansyah and J.C.A.M. Buijs. Mining process performance from event logs: the BPI Challenge 2012 case study. Technical report, BPM Center Report, No. BPM-12-15, 2012

- M. de Leoni, J.C.A.M. Buijs, W.M.P. van der Aalst, and A.H.M. ter Hofstede. Facilitating process analysis through visualising process history: Experiences with a dutch municipality

SIKS dissertations

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), *Operating 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 (UvT), *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.*

2010

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.*
2010-48 Withdrawn, .
2010-49 Jahn-Takeshi Saito (UM), *Solving difficult game positions.*
2010-50 Bouke Huurnink (UVA), *Search in Audiovisual Broadcast Archives.*
2010-51 Alia Khairia Amin (CWI), *Understanding and supporting information seeking tasks in multiple sources.*
2010-52 Peter-Paul van Maanen (VU), *Adaptive Support for Human-Computer Teams: Exploring the Use of Cognitive Models of Trust and Attention.*
2010-53 Edgar Meij (UVA), *Combining Concepts and Language Models for Information Access.*

2011

2011-01 Botond Cseke (RUN), *Variational Algorithms for Bayesian Inference in Latent Gaussian Models.*
2011-02 Nick Tinnemeier(UU), *Organizing Agent Organizations. Syntax and Operational Semantics of an Organization-Oriented Programming Language.*
2011-03 Jan Martijn van der Werf (TUE), *Compositional Design and Verification of Component-Based Information Systems.*
2011-04 Hado van Hasselt (UU), *Insights in Reinforcement Learning; Formal analysis and empirical evaluation of temporal-difference.*
2011-05 Base van der Raadt (VU), *Enterprise Architecture Coming of Age - Increasing the Performance of an Emerging Discipline..*
2011-06 Yiwen Wang (TUE), *Semantically-Enhanced Recommendations in Cultural Heritage.*
2011-07 Yujia Cao (UT), *Multimodal Information Presentation for High Load Human Computer Interaction.*
2011-08 Nieske Vergunst (UU), *BDI-based Generation of Robust Task-Oriented Dialogues.*
2011-09 Tim de Jong (OU), *Contextualised Mobile Media for Learning.*
2011-10 Bart Bogaert (UvT), *Cloud Content Contention.*
2011-11 Dhaval Vyas (UT), *Designing for Awareness: An Experience-focused HCI Perspective.*
2011-12 Carmen Bratosin (TUE), *Grid Architecture for Distributed Process Mining.*
2011-13 Xiaoyu Mao (UvT), *Airport under Control. Multiagent Scheduling for Airport Ground Handling.*
2011-14 Milan Lovric (EUR), *Behavioral Finance and Agent-Based Artificial Markets.*
2011-15 Marijn Koolen (UvA), *The Meaning of Structure: the Value of Link Evidence for Information Retrieval.*
2011-16 Maarten Schadd (UM), *Selective Search in Games of Different Complexity.*
2011-17 Jiyin He (UVA), *Exploring Topic Structure: Coherence, Diversity and Relatedness.*
2011-18 Mark Ponsen (UM), *Strategic Decision-Making in complex games.*
2011-19 Ellen Rusman (OU), *The Mind ' s Eye on Personal Profiles.*
2011-20 Qing Gu (VU), *Guiding service-oriented software engineering - A view-based approach.*
2011-21 Linda Terlouw (TUD), *Modularization and Specification of Service-Oriented*

Systems.
2011-22 Junte Zhang (UVA), *System Evaluation of Archival Description and Access.*
2011-23 Wouter Weerkamp (UVA), *Finding People and their Utterances in Social Media.*
2011-24 Herwin van Welbergen (UT), *Behavior Generation for Interpersonal Coordination with Virtual Humans On Specifying, Scheduling and Realizing Multimodal Virtual Human Behavior.*
2011-25 Syed Waqar ul Qounain Jaffry (VU)), *Analysis and Validation of Models for Trust Dynamics.*
2011-26 Matthijs Aart Pontier (VU), *Virtual Agents for Human Communication - Emotion Regulation and Involvement-Distance Trade-Offs in Embodied Conversational Agents and Robots.*
2011-27 Aniel Bhulai (VU), *Dynamic website optimization through autonomous management of design patterns.*
2011-28 Rianne Kaptein(UVA), *Effective Focused Retrieval by Exploiting Query Context and Document Structure.*
2011-29 Faisal Kamiran (TUE), *Discrimination-aware Classification.*
2011-30 Egon van den Broek (UT), *Affective Signal Processing (ASP): Unraveling the mystery of emotions.*
2011-31 Ludo Waltman (EUR), *Computational and Game-Theoretic Approaches for Modeling Bounded Rationality.*
2011-32 Nees-Jan van Eck (EUR), *Methodological Advances in Bibliometric Mapping of Science.*
2011-33 Tom van der Weide (UU), *Arguing to Motivate Decisions.*
2011-34 Paolo Turrini (UU), *Strategic Reasoning in Interdependence: Logical and Game-theoretical Investigations.*
2011-35 Maaike Harbers (UU), *Explaining Agent Behavior in Virtual Training.*
2011-36 Erik van der Spek (UU), *Experiments in serious game design: a cognitive approach.*
2011-37 Adriana Burlutiu (RUN), *Machine Learning for Pairwise Data, Applications for Preference Learning and Supervised Network Inference.*
2011-38 Nyree Lemmens (UM), *Bee-inspired Distributed Optimization.*
2011-39 Joost Westra (UU), *Organizing Adaptation using Agents in Serious Games.*
2011-40 Viktor Clerc (VU), *Architectural Knowledge Management in Global Software Development.*
2011-41 Luan Ibraimi (UT), *Cryptographically Enforced Distributed Data Access Control.*
2011-42 Michal Sindlar (UU), *Explaining Behavior through Mental State Attri-*

bution.
2011-43 Henk van der Schuur (UU), *Process Improvement through Software Operation Knowledge.*
2011-44 Boris Reuderink (UT), *Robust Brain-Computer Interfaces.*
2011-45 Herman Stehouwer (UvT), *Statistical Language Models for Alternative Sequence Selection.*
2011-46 Beibei Hu (TUD), *Towards Contextualized Information Delivery: A Rule-based Architecture for the Domain of Mobile Police Work.*
2011-47 Azizi Bin Ab Aziz(VU), *Exploring Computational Models for Intelligent Support of Persons with Depression.*
2011-48 Mark Ter Maat (UT), *Response Selection and Turn-taking for a Sensitive Artificial Listening Agent.*
2011-49 Andreea Niculescu (UT), *Conversational interfaces for task-oriented spoken dialogues: design aspects influencing interaction quality.*

2012

2012-01 Terry Kakeeto (UvT), *Relationship Marketing for SMEs in Uganda.*
2012-02 Muhammad Umair(VU), *Adaptivity, emotion, and Rationality in Human and Ambient Agent Models.*
2012-03 Adam Vanya (VU), *Supporting Architecture Evolution by Mining Software Repositories.*
2012-04 Jurriaan Souer (UU), *Development of Content Management System-based Web Applications.*
2012-05 Marijn Plomp (UU), *Maturing Interorganisational Information Systems.*
2012-06 Wolfgang Reinhardt (OU), *Awareness Support for Knowledge Workers in Research Networks.*
2012-07 Rianne van Lambalgen (VU), *When the Going Gets Tough: Exploring Agent-based Models of Human Performance under Demanding Conditions.*
2012-08 Gerben de Vries (UVA), *Kernel Methods for Vessel Trajectories.*
2012-09 Ricardo Neisse (UT), *Trust and Privacy Management Support for Context-Aware Service Platforms.*
2012-10 David Smits (TUE), *Towards a Generic Distributed Adaptive Hypermedia Environment.*
2012-11 J.C.B. Rantham Prabhakara (TUE), *Process Mining in the Large: Preprocessing, Discovery, and Diagnostics.*
2012-12 Kees van der Sluijs (TUE), *Model Driven Design and Data Integration*

in Semantic Web Information Systems.
2012-13 Suleman Shahid (UvT), *Fun and Face: Exploring non-verbal expressions of emotion during playful interactions.*
2012-14 Evgeny Knutov(TUE), *Generic Adaptation Framework for Unifying Adaptive Web-based Systems.*
2012-15 Natalie van der Wal (VU), *Social Agents. Agent-Based Modelling of Integrated Internal and Social Dynamics of Cognitive and Affective Processes..*
2012-16 Fiemke Both (VU), *Helping people by understanding them - Ambient Agents supporting task execution and depression treatment.*
2012-17 Amal Elgammal (UvT), *Towards a Comprehensive Framework for Business Process Compliance.*
2012-18 Eltjo Poort (VU), *Improving Solution Architecting Practices.*
2012-19 Helen Schonenberg (TUE), *What's Next? Operational Support for Business Process Execution.*
2012-20 Ali Bahramisharif (RUN), *Covert Visual Spatial Attention, a Robust Paradigm for Brain-Computer Interfacing.*
2012-21 Roberto Cornacchia (TUD), *Querying Sparse Matrices for Information Retrieval.*
2012-22 Thijs Vis (UvT), *Intelligence, politie en veiligheidsdienst: verenigbare grootheden?.*
2012-23 Christian Muehl (UT), *Toward Affective Brain-Computer Interfaces: Exploring the Neurophysiology of Affect during Human Media Interaction.*
2012-24 Laurens van der Werff (UT), *Evaluation of Noisy Transcripts for Spoken Document Retrieval.*
2012-25 Silja Eckartz (UT), *Managing the Business Case Development in Inter-Organizational IT Projects: A Methodology and its Application.*
2012-26 Emile de Maat (UVA), *Making Sense of Legal Text.*
2012-27 Hayrettin Gurkok (UT), *Mind the Sheep! User Experience Evaluation & Brain-Computer Interface Games.*
2012-28 Nancy Pascall (UvT), *Engendering Technology Empowering Women.*
2012-29 Almer Tigelaar (UT), *Peer-to-Peer Information Retrieval.*
2012-30 Alina Pommeranz (TUD), *Designing Human-Centered Systems for Reflective Decision Making.*
2012-31 Emily Bagarukayo (RUN), *A Learning by Construction Approach for Higher Order Cognitive Skills Improvement, Building Capacity and Infrastructure.*
2012-32 Wietske Visser (TUD), *Qualitative multi-criteria preference representation and reasoning.*
2012-33 Rory Sie (OUN), *Coalitions in Cooperation Networks (COCOON).*
2012-34 Pavol Jancura (RUN), *Evolutionary analysis in PPI networks and appli-*

cations.

2012-35 Evert Haasdijk (VU), *Never Too Old To Learn – On-line Evolution of Controllers in Swarm- and Modular Robotics.*

2012-36 Denis Ssebugwawo (RUN), *Analysis and Evaluation of Collaborative Modeling Processes.*

2012-37 Agnes Nakakawa (RUN), *A Collaboration Process for Enterprise Architecture Creation.*

2012-38 Selmar Smit (VU), *Parameter Tuning and Scientific Testing in Evolutionary Algorithms.*

2012-39 Hassan Fatemi (UT), *Risk-aware design of value and coordination networks.*

2012-40 Agus Gunawan (UvT), *Information Access for SMEs in Indonesia.*

2012-41 Sebastian Kelle (OU), *Game Design Patterns for Learning.*

2012-42 Dominique Verpoorten (OU), *Reflection Amplifiers in self-regulated Learning.*

2012-43 Withdrawn, .

2012-44 Anna Tordai (VU), *On Combining Alignment Techniques.*

2012-45 Benedikt Kratz (UvT), *A Model and Language for Business-aware Transactions.*

2012-46 Simon Carter (UVA), *Exploration and Exploitation of Multilingual Data for Statistical Machine Translation.*

2012-47 Manos Tsagkias (UVA), *Mining Social Media: Tracking Content and Predicting Behavior.*

2012-48 Jorn Bakker (TUE), *Handling Abrupt Changes in Evolving Time-series Data.*

2012-49 Michael Kaisers (UM), *Learning against Learning - Evolutionary dynamics of reinforcement learning algorithms in strategic interactions.*

2012-50 Steven van Kervel (TUD), *Ontologogy driven Enterprise Information Systems Engineering.*

2012-51 Jeroen de Jong (TUD), *Heuristics in Dynamic Sceduling; a practical framework with a case study in elevator dispatching.*

2013

2013-01 Viorel Milea (EUR), *News Analytics for Financial Decision Support.*

2013-02 Erietta Liarou (CWI), *MonetDB/DataCell: Leveraging the Column-store Database Technology for Efficient and Scalable Stream Processing.*

2013-03 Szymon Klarman (VU), *Reasoning with Contexts in Description Logics.*
2013-04 Chetan Yadati(TUD), *Coordinating autonomous planning and scheduling.*
2013-05 Dulce Pumareja (UT), *Groupware Requirements Evolutions Patterns.*
2013-06 Romulo Goncalves(CWI), *The Data Cyclotron: Juggling Data and Queries for a Data Warehouse Audience.*
2013-07 Giel van Lankveld (UvT), *Quantifying Individual Player Differences.*
2013-08 Robbert-Jan Merk(VU), *Making enemies: cognitive modeling for opponent agents in fighter pilot simulators.*
2013-09 Fabio Gori (RUN), *Metagenomic Data Analysis: Computational Methods and Applications.*
2013-10 Jeewanie Jayasinghe Arachchige(UvT), *A Unified Modeling Framework for Service Design..*
2013-11 Evangelos Pournaras(TUD), *Multi-level Reconfigurable Self-organization in Overlay Services.*
2013-12 Marian Razavian(VU), *Knowledge-driven Migration to Services.*
2013-13 Mohammad Safiri(UT), *Service Tailoring: User-centric creation of integrated IT-based homecare services to support independent living of elderly.*
2013-14 Jafar Tanha (UVA), *Ensemble Approaches to Semi-Supervised Learning Learning.*
2013-15 Daniel Hennes (UM), *Multiagent Learning - Dynamic Games and Applications.*
2013-16 Eric Kok (UU), *Exploring the practical benefits of argumentation in multi-agent deliberation.*
2013-17 Koen Kok (VU), *The PowerMatcher: Smart Coordination for the Smart Electricity Grid.*
2013-18 Jeroen Janssens (UvT), *Outlier Selection and One-Class Classification.*
2013-19 Renze Steenhuizen (TUD), *Coordinated Multi-Agent Planning and Scheduling.*
2013-20 Katja Hofmann (UvA), *Fast and Reliable Online Learning to Rank for Information Retrieval.*
2013-21 Sander Wubben (UvT), *Text-to-text generation by monolingual machine translation.*
2013-22 Tom Claassen (RUN), *Causal Discovery and Logic.*
2013-23 Patricio de Alencar Silva(UvT), *Value Activity Monitoring.*
2013-24 Haitham Bou Ammar (UM), *Automated Transfer in Reinforcement Learning.*
2013-25 Agnieszka Anna Latoszek-Berendsen (UM), *Intention-based Decision Support. A new way of representing and implementing clinical guidelines in a De-*

cision Support System.

2013-26 Alireza Zarghami (UT), *Architectural Support for Dynamic Homecare Service Provisioning.*

2013-27 Mohammad Huq (UT), *Inference-based Framework Managing Data Provenance.*

2013-28 Frans van der Sluis (UT), *When Complexity becomes Interesting: An Inquiry into the Information eXperience.*

2013-29 Iwan de Kok (UT), *Listening Heads.*

2013-30 Joyce Nakatumba (TUE), *Resource-Aware Business Process Management: Analysis and Support.*

2013-31 Dinh Khoa Nguyen (UvT), *Blueprint Model and Language for Engineering Cloud Applications.*

2013-32 Kamakshi Rajagopal (OUN), *Networking For Learning; The role of Networking in a Lifelong Learner's Professional Development.*

2013-33 Qi Gao (TUD), *User Modeling and Personalization in the Microblogging Sphere.*

2013-34 Kien Tjin-Kam-Jet (UT), *Distributed Deep Web Search.*

2013-35 Abdallah El Ali (UvA), *Minimal Mobile Human Computer Interaction.*

2013-36 Than Lam Hoang (TUe), *Pattern Mining in Data Streams.*

2013-37 Dirk Börner (OUN), *Ambient Learning Displays.*

2013-38 Eelco den Heijer (VU), *Autonomous Evolutionary Art.*

2013-39 Joop de Jong (TUD), *A Method for Enterprise Ontology based Design of Enterprise Information Systems.*

2013-40 Pim Nijssen (UM), *Monte-Carlo Tree Search for Multi-Player Games.*

2013-41 Jochem Liem (UVA), *Supporting the Conceptual Modelling of Dynamic Systems: A Knowledge Engineering Perspective on Qualitative Reasoning.*

2013-42 Léon Planken (TUD), *Algorithms for Simple Temporal Reasoning.*

2013-43 Marc Bron (UVA), *Exploration and Contextualization through Interaction and Concepts.*

2014

2014-01 Nicola Barile (UU), *Studies in Learning Monotone Models from Data.*

2014-02 Fiona Tuliyano (RUN), *Combining System Dynamics with a Domain Modeling Method.*

2014-03 Sergio Raul Duarte Torres (UT), *Information Retrieval for Children: Search Behavior and Solutions.*

2014-04 Hanna Jochmann-Mannak (UT), *Websites for children: search strategies and interface design - Three studies on children's search performance and evaluation.*

2014-05 Jurriaan van Reijsen (UU), *Knowledge Perspectives on Advancing Dynamic Capability.*

2014-06 Damian Tamburri (VU), *Supporting Networked Software Development.*

2014-07 Arya Adriansyah (TUE), *Aligning Observed and Modeled Behavior.*

2014-08 Samur Araujo (TUD), *Data Integration over Distributed and Heterogeneous Data Endpoints.*

2014-09 Philip Jackson (UvT), *Toward Human-Level Artificial Intelligence: Representation and Computation of Meaning in Natural Language.*

2014-10 Ivan Salvador Razo Zapata (VU), *Service Value Networks.*

2014-11 Janneke van der Zwaan (TUD), *An Empathic Virtual Buddy for Social Support.*

2014-12 Willem van Willigen (VU), *Look Ma, No Hands: Aspects of Autonomous Vehicle Control.*

2014-13 Arlette van Wissen (VU), *Agent-Based Support for Behavior Change: Models and Applications in Health and Safety Domains.*

2014-14 Yangyang Shi (TUD), *Language Models With Meta-information.*

2014-15 Natalya Mogles (VU), *Agent-Based Analysis and Support of Human Functioning in Complex Socio-Technical Systems: Applications in Safety and Healthcare.*

2014-16 Krystyna Milian (VU), *Supporting trial recruitment and design by automatically interpreting eligibility criteria.*

2014-17 Kathrin Dentler (VU), *Computing healthcare quality indicators automatically: Secondary Use of Patient Data and Semantic Interoperability.*

2014-18 Mattijs Ghijsen (VU), *Methods and Models for the Design and Study of Dynamic Agent Organizations.*

2014-19 Vincius Ramos (TUE), *Adaptive Hypermedia Courses: Qualitative and Quantitative Evaluation and Tool Support.*

2014-20 Mena Habib (UT), *Named Entity Extraction and Disambiguation for Informal Text: The Missing Link.*

2014-21 Kassidy Clark (TUD), *Negotiation and Monitoring in Open Environments.*

2014-22 Marieke Peeters (UT), *Personalized Educational Games - Developing agent-supported scenario-based training.*

2014-23 Eleftherios Sidirourgos (UvA/CWI), *Space Efficient Indexes for the Big Data Era.*

2014-24 Davide Ceolin (VU), *Trusting Semi-structured Web Data.*

2014-25 Martijn Lappenschaar (RUN), *New network models for the analysis of disease interaction.*
2014-26 Tim Baarslag (TUD), *What to Bid and When to Stop.*
2014-27 Rui Jorge Almeida (EUR), *Conditional Density Models Integrating Fuzzy and Probabilistic Representations of Uncertainty.*
2014-28 Anna Chmielowiec (VU), *Decentralized k-Clique Matching.*
2014-29 Jaap Kabbedijk (UU), *Variability in Multi-Tenant Enterprise Software.*
2014-30 Peter de Kock Berenschot (UvT), *Anticipating Criminal Behaviour.*
2014-31 Leo van Moergestel (UU), *Agent Technology in Agile Multiparallel Manufacturing and Product Support.*
2014-32 Naser Ayat (UVA), *On Entity Resolution in Probabilistic Data.*
2014-33 Tesfa Tegegne Asfaw (RUN), *Service Discovery in eHealth.*
2014-34 Christina Manteli (VU), *The Effect of Governance in Global Software Development: Analyzing Transactive Memory Systems.*
2014-35 Joost van Oijen (UU), *Cognitive Agents in Virtual Worlds: A Middleware Design Approach.*
2014-36 Joos Buijs (TUE), *Flexible Evolutionary Algorithms for Mining Structured Process Models.*