

**School of Electrical Engineering and Computing
Department of Computing**

**Mining Complex Structured Data:
Enhanced Methods and Applications**

Dang Bach Bui

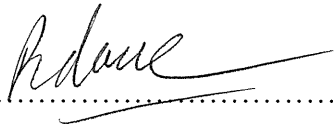
**This thesis is presented for the Degree of
Doctor of Philosophy
of
Curtin University**

May 2015

Declaration

To the best of my knowledge and belief this thesis contains no material previously published by any other person except where due acknowledgment has been made.

This thesis contains no material which has been accepted for the award of any other degree or diploma in any university.

Signature: 

Date: ..01./05/2015

Abstract

This thesis offers an alternative approach to mining business process event log, which has been mostly conducted via process models. The approach was motivated by the increasing availability of *XML*-based event logs and *XML* mining methods. An integrated method, called *PCFSM*, which allows for direct applications of a wide range of relational data mining techniques, e.g. decision tree learning, association rule mining, to *XML*-based event data, was proposed. The advantages of this method are that the structural complexities of *XML* documents are reduced and the patterns found are more informative than with traditional methods.

An exploratory analysis method was also introduced to analyse process logs in an unbiased way where there is a lack of domain knowledge to guide the analysis process. The main point of this method is to apply unsupervised learning algorithms to the table representation of the tree-structured process log, then similar process instances can be grouped together for further analysis. Relational data mining techniques, such as frequent itemset mining and decision tree learning, can be utilised to discover the common or distinguishing characteristics of each cluster of instances.

To predict the outcomes of process instances according to various business goals, an associative classifier, called *DSMC*, which is an extension of *PCFSM*, was proposed. Experiments showed that this method is comparable to other traditional subtree mining methods in terms of accuracy rates. Although *DSMC*'s coverage rates are often lower than traditional approaches for the same minimum support, it is able to operate at a much lower support threshold.

The proposed methods were evaluated on synthetic, real-world process logs and other types of tree-structured or *XML* data. The experimental results showed that our approach is able to provide valuable insights into event data. Furthermore, the two suggested methods are able to predict whether running process instances complete satisfying a predefined outcome, or to recommend possible actions to achieve that outcome.

Overall, this work extends the available pool of process analysis techniques, allowing efficient knowledge discovery from *XML*-based process logs in a more direct, unbiased manner, and provides an effective associative classifier for event data.

Acknowledgments

The long and winding road has nearly come to an end. Along the way I have met lovely people and made many friends, who have been sources of inspiration and encouragement during this memorable but difficult journey.

First of all, I would like to express my deepest thanks to my supervisor, Dr Fedja Hadzic. The feedback and guidance he gave to me always contained a great responsibility and sharpness. He is also a good friend, who has given me lots of encouragement. I also want to thank him for bringing me to the world of data mining, which is full of joy.

I have been fortunate to have the opportunity to work with my co-supervisor, Dr Michael Hecker, who is one of the best techies I have known. His technical and programming skills are amazing. Without his help, I would have drowned in a sea of code. Thank you so much, Michael.

It was my privilege and honour to have Dr Duc-Son Pham as my supervisor in the last phase of my study. He has given me considerable help. Without him, I would have been in big trouble. Thank you very much.

My journey would not even have happened if Professor Robert Meersman had not introduced me to this place. The support you have given me over such a long time has boosted my confidence. I owe you a special debt of thanks.

I gratefully acknowledge the financial support of Curtin University and VIED for this project.

I am grateful to Associate Professor Mihai Lazarescu and Ms Mary Mulligan for their administrative . My thanks also go to Vidyasagar Potdar, my previous supervisor, who was also very supportive of me. It has been my pleasure to work with Dr Kit Yan Chan. The lunchtimes seemed to pass very quickly when talking to him. Special thanks go to you, Kit.

The time I was in EU4 was short but it was memorable because I had many nice and loyal friends. I would like to thank Novita, my best friend. It is hard to find a friend to be there whenever you need them. Novitas inner strength has also inspired me a lot. Ilham and Nee have been my close friends for a long time and will remain so for many years to come. I will definitely remember the warmth and laughs we had during our many lunch trips.

I am grateful to my friends in building 314. Thanks Viet; your sense of humour is superb, and the laughs we had together reduced my stress. Thanks Chitra; we had many interesting talks. Thanks, Quang, for your great help. Many thanks to chi Linh, Yuthika, anh Hung, chi Thanh and Jeff; we had a lot of laughs and I will miss spending lunchtimes together.

There are many other people who have helped me in one or another way during this voyage, and I would like to thank them all.

I would like to give my deepest thanks to my parents, who have given me tremendous encouragement and support during these last years. I especially want to thank my parents-in-law for their constant support. The last few hectic months spent writing my thesis would have driven me crazy without the help of my wife's auntie. Thank you, auntie.

Last, but not least, I would like to express my deep appreciation to my dear wife for her care, love and tolerance. Your unflagging support has enabled me to fly high. Thanks Annie, my little baby, for being there, beautifully.

Contents

Abstract	ii
Acknowledgments	iii
List of Publications	x
List of Figures	xi
List of Tables	xv
Acronyms	xix
1 Introduction	1
1.1 Business Process Management	2
1.2 Process Mining	4
1.3 Motivation	6
1.3.1 Process Mining without Process Models	6
1.3.2 The Advent of <i>XML</i> -based Event Logs	8
1.3.3 Examples of <i>XML</i> Mining Methods and Their Applications . .	11
1.4 Research Questions and Objectives	15
1.5 Research Contributions and Significance	16
1.6 Thesis Outline	17
2 Literature Review	19
2.1 Business Process Management	19
2.1.1 History of <i>Business Process Management (BPM)</i>	19
2.1.2 Process Model	20
2.1.3 Business Process Identification	22
2.1.4 Process Design	22

2.1.5	Process Analysis - Qualitative Methods	23
2.1.6	Process Analysis - Quantitative Methods	23
2.1.7	Process Redesign	24
2.1.8	Business Process Implementation/Configurations	25
2.1.9	Process Enactment	26
2.2	Process Mining	26
2.2.1	Event Logs	29
2.2.2	Event Log Structure	29
2.2.3	Notation	30
2.2.4	XES	31
2.2.5	Process Model Discovery	33
2.2.6	Main Approaches to Process Log Analysis	36
2.3	Related Data Mining Concepts	38
2.3.1	Types of Data	39
2.3.2	Frequent Pattern Mining	41
2.3.3	Frequent Itemset Mining	42
2.3.4	Frequent Subsequence Mining	43
2.3.5	Frequent Subtree Mining	44
2.3.6	Frequent Subgraph Mining	44
2.3.7	Closed and Maximal Patterns	44
2.3.8	Frequent Pattern Mining Approaches	45
2.3.9	Association Rule Mining	46
2.3.10	Classification	47
2.3.11	Classification Evaluation	49
2.3.12	Obtaining Reliable Accuracy Estimates	51
2.3.13	Associative Classification	53
2.3.14	Interestingness	53
2.4	<i>XML</i> Data Mining Methods	55
2.4.1	The Relation between <i>XML</i> and Tree-structured Data	56
2.4.2	Formalism of Tree-structured Data	57
2.4.3	Frequent Subtree Mining	60
2.4.4	Frequent Subtree Mining Algorithms	61
2.5	Structural <i>XML</i> Classification	63

2.6	Summary and Research Gaps	66
3	PCFSM: A New Method for Mining Process Logs	69
3.1	Mining Process Logs Using Tree-based Techniques	70
3.1.1	A Motivating Example	70
3.1.2	Postion-constrained Subtrees	72
3.2	The Structure-preserving Tree Mining Approach	73
3.2.1	<i>Database Structure Model (DSM)</i> extraction	74
3.2.2	Conversion to Flat Data Format	83
3.2.3	Data Mining Methods	88
3.2.4	Illustration of the Main Concepts	90
3.2.5	<i>DSM</i> with Minimum Support	91
3.3	Time Performance of <i>DSM</i> -based Frequent Subtree Mining and Traditional Methods	94
3.4	The Proposed Methods	97
3.4.1	The Position-constrained Tree-structured Mining Approach for Process Log Analysis (<i>PCFSM</i>)	98
3.4.2	Exploratory Process Log Analysis Method	103
3.5	Discussion	104
4	Evaluation of PCFSM on Process Logs	107
4.1	Experimental Settings	107
4.1.1	Hospital Dataset	107
4.1.2	Teleclaim Dataset	108
4.1.3	Telephone Repair Dataset	108
4.1.4	Summary on Datasets	109
4.2	Evaluation of the <i>EPLA</i> method	109
4.2.1	Discovering Process Execution Groups	109
4.2.2	Discovering Descriptive Characteristics of Groups	110
4.2.3	Discovering Discriminating Characteristics of Groups	112
4.3	Classification based on <i>PCFSM</i> on Labelled Dataset	115
4.4	Lessons Learned and General Remarks	117
5	A Position-constrained Associative Classification Approach	121
5.1	Associative Classification on Process Logs	121

5.1.1	Frequent Subtree-based Classification	122
5.1.2	<i>DSMC</i> —the Proposed Classification Method	124
5.1.3	Illustration of the Proposed Classification Method	125
5.2	Experimental Settings	131
5.3	Experimental Results on Structurally Heterogeneous Data	134
5.3.1	General Comparison	134
5.3.2	Analysis of the Rule Set	140
5.3.3	Comparison on Imbalanced Classes	146
5.3.4	Synthetic Datasets	148
5.4	Experimental Results on Structurally Homogeneous Data	149
5.4.1	Hospital Dataset	150
5.4.2	<i>CRM</i> Dataset	150
5.5	Predicting Outcomes of Running Process Instances	152
5.5.1	Hospital Dataset	152
5.5.2	Dutch Financial Institute Dataset	155
5.5.3	Telephone Repair Dataset	157
5.5.4	Summary	158
5.6	Recommendation Model for Running Process Instances	158
5.6.1	Recommendation Model for the Hospital Dataset	160
5.6.2	Recommendation Model for the <i>Dutch Financial Institute (DFI)</i> Dataset	161
5.6.3	Recommendation Model for the Telephone Repair Dataset . .	162
5.6.4	Summary	162
5.7	Enhancing <i>DSMC</i>	163
5.8	Conclusion	164
6	Conclusion and Future Work	171
6.1	Recapitulation	171
6.2	Contributions	172
6.3	Limitations	174
6.4	Future Work	175
6.4.1	Efficient User Interaction	175
6.4.2	Combination of <i>DSMC</i> Rules and Traditional Frequent Sub- tree based Rules	176

6.4.3 Outlier Detection	176
A Experimental Results on CSLOG	178
Index	185
Bibliography	187

List of Publications

The following papers have been published either based on or related to the research findings presented in this thesis:

- Bui, D. B., Hadzic, F., and Potdar, V. (2012b). A Framework for Application of Tree-Structured Data Mining to Process Log Analysis. In Yin, H., Costa, J. A., and Barreto, G., editors, *Proceedings of the 13th International Conference on Intelligent Data Engineering and Automated Learning*, volume 7435 of *Lecture Notes in Computer Science*, pages 424–434. Springer.
- Bui, D. B., Hadzic, F., and Hecker, M. (2012a). Application of Tree-structured Data Mining for Analysis of Process Logs in XML format. In Zhao, Y., Li, J., Kennedy, P. J., and Christen, P., editors, *Proceedings of the 10th Australasian Data Mining Conference (AusDM 2012)*, pages 109–118, Sydney, Australia. ACS.
- Bui, D. B., Hadzic, F., and Hecker, M. (2013). Evaluation of Position-Constrained Association-Rule-Based Classification for Tree-Structured Data. In Li, J., Cao, L., Wang, C., Tan, K. C., Liu, B., Pei, J., and Tseng, V. S., editors, *Trends and Applications in Knowledge Discovery and Data Mining*, volume 7867 of *Lecture Notes in Computer Science*, pages 379–391. Springer Berlin Heidelberg.
- Bui, D. B., Hadzic, F., Tagarelli, A., and Hecker, M. (2014). Evaluation of an Associative Classifier based on Position-constrained Frequent/closed Subtree Mining. *Journal of Intelligent Information Systems*, 42(2):1–25.
- Bui, D. B., Hadzic, F., and Hecker, M. (2015). Direct and Unbiased Analysis of XML Process Logs Using Position and Structure Aware Data Mining. *Journal of Research and Practice in Information Technology*, 46(4). (to appear).

List of Figures

1.1	A <i>BPM</i> life cycle.	2
1.2	A process mining architecture.	5
1.3	An <i>XML</i> mining taxonomy (Nayak, 2008).	11
1.4	A subtree pattern in different contexts.	11
1.5	Tree database T_{HOS} represents the event log shown in Fig. 1.1.	12
1.6	Two frequent subtrees found from T_{HOS}	13
1.7	Two association rules found from T_{HOS}	14
1.8	(a) A rule-based prediction model learned from the tree database in Fig. 1.5. (b) A test process instance.	15
2.1	An example of process model in Petri net notation.	21
2.2	<i>XES</i> meta model (Verbeek et al., 2011).	31
2.3	An example of a decision tree that predicts species of an iris.	48
2.4	S_1 is an induced subtree of T	59
2.5	S_2 is an embedded subtree of T	59
2.6	Position of vertices in a tree.	61
3.1	T_{HOS} —the hospital database (reproduced from Chapter 1 for convenience).	71
3.2	Patterns learned from Heuristic Miner.	72
3.3	Two frequent closed subtrees of T_{HOS} with support=2.	72
3.4	The structure-preserving tree mining approach.	74
3.5	Tree-structured database T_{EX}	75
3.6	An example of top-left subtree relation.	76
3.7	The <i>DSM</i> of T_{EX}	79
3.8	The top-left mirror of tree instances on the <i>DSM</i> of T_{EX}	80
3.9	Each node and edge of the <i>DSM</i> of T_{EX} is named by their position and backtracking order.	81
3.10	An illustration for the proof of 3.11.	81

3.11	The position-constrained top-left mirrors (shaded areas) of trees on the <i>DSM</i>	85
3.12	The reconstruction of $pFPCT = \{aX_0, bX_6, cX_7\}$	89
3.13	The reconstruction of $pFPCT = \{bX_2, cX_5, aX_6, dX_7, eX_{10}\}$	90
3.14	An example of Property 3.16.	90
3.15	An example of Property 3.17.	91
3.16	Illustration of the main concepts in the structure-preserving tree mining approach.	93
3.17	The <i>DSM</i> of T_{EX} with a support threshold of 40% is shown in (a), and the mapping of tree instances of T_{EX} to the <i>DSM</i> are shown in (b), (c), (d), (e), and (f).	94
3.18	Time performance of different methods.	96
3.19	Number of patterns of different methods.	96
3.20	The <i>PCFSM</i> method.	99
3.21	T_{HOS} with integer values.	101
3.22	The <i>DSM</i> of T_{HOS}	102
3.23	A reconstructed position-constrained pattern of T_{HOS}	103
3.24	The <i>EPLA</i> method.	104
4.1	A frequent subtree in cluster 0.	112
4.2	A frequent subtree in cluster 1.	112
4.3	A position-constrained frequent subtree in cluster 0.	113
4.4	A position-constrained frequent subtree in cluster 1.	113
4.5	A decision tree learned from three clusters of process instances. . . .	114
4.6	Urgency prediction of a process instance on diagnosis code <i>M13</i> and treatment code <i>803</i>	115
4.7	Urgency prediction of a process instance on diagnosis code <i>M11</i>	116
4.8	A decision tree for outcomes of an insurance claim in the <i>teleclaim</i> dataset.	117
4.9	A decision tree for complexity prediction of the <i>telephone repair</i> dataset. .	118
5.1	An illustration for classification evaluation.	124
5.2	Tree database T_{cl}	125
5.3	Four frequent embedded subtrees of T_{cl} and their corresponding supports. .	126
5.4	T_{eval} —a synthetic tree database used for classification evaluation. . . .	127
5.5	The <i>DSM</i> tree of T_{cl}	128
5.6	The <i>DSM</i> tree at minimum support = 50% of T_{cl}	128

5.7	Four position-constrained embedded subtrees of T_{cl} and their corresponding supports and confidences.	129
5.8	Accuracy rates of dataset <i>CSI-2</i> where $(\mathfrak{w})\mathfrak{c}=50\%$ or $l=1$	135
5.9	Accuracy rates of dataset <i>CSI-2</i> where $(\mathfrak{w})\mathfrak{c}=90\%$ or $l=10$	136
5.10	Coverage rates of dataset <i>CSI-2</i> where $(\mathfrak{w})\mathfrak{c}=50\%$ or $l=1$	137
5.11	Coverage rates of dataset <i>CSI-2</i> where $(\mathfrak{w})\mathfrak{c}=90\%$ or $l=10$	138
5.12	Comparison of accuracy rates based on frequent pattern types in the <i>CSI-2</i> dataset.	140
5.13	Comparison of coverage rates based on frequent pattern types in the <i>CSI-2</i> dataset.	141
5.14	Comparison of accuracy rates based on subtree types in the <i>CSI-2</i> dataset.	142
5.15	Comparison of coverage rates based on subtree types in the <i>CSI-2</i> dataset.	143
5.16	Comparison of accuracy rates based on rule strength measures in the <i>CSI-2</i> dataset.	144
5.17	Comparison of coverage rates based on rule strength measures in the <i>CSI-2</i> dataset.	145
5.18	The number of instances which one method predicts correctly whereas the other wrong (based on similar rules).	146
5.19	Accuracy and coverage rates of synthetic datasets with $\mathfrak{s} = 1\%$	148
5.20	Accuracy and coverage rates of synthetic datasets with $\mathfrak{s} = 2\%$	149
5.21	A (partial) recommendation model for a testing instance–Hospital dataset.	160
5.22	A (partial) recommendation model for a testing instance– <i>DFI</i> dataset.	161
5.23	A (partial) recommendation model for a testing instance–Telephone Repair dataset.	162
5.24	Accuracy rates of <i>DSMC</i> using the stable-rule-set and normal method on the <i>CSI-2</i> dataset.	164
5.25	Coverage rates of <i>DSMC</i> using the stable-rule-set and normal method on the <i>CSI-2</i> dataset.	165
5.26	Accuracy rates of <i>DSMC</i> using the stable-rule-set and normal method on the hospital dataset.	166
5.27	Accuracy rates of <i>DSMC</i> using the stable-rule-set and normal method on <i>CRM</i> dataset.	167
5.28	Coverage rates of <i>DSMC</i> using the stable-rule-set and normal method on <i>CRM</i> dataset.	168

5.29	The number of rules of the stable-rule-set and normal methods on the <i>CSI-2</i> dataset.	168
5.30	The number of rules of the stable-rule-set and normal methods on the hospital dataset.	169
5.31	The number of rules of the stable-rule-set and normal methods on the <i>CRM</i> dataset.	169
5.32	The effect of <i>ADABoosting</i> on the accuracy rates of <i>DSMC</i> on <i>CSI-2</i>	170
5.33	The effect of <i>ADABoosting</i> on the coverage rates of <i>DSMC</i> on <i>CSI-2</i>	170

List of Tables

2.1	An event log for a conference reviewing process.	30
2.2	Categorisation of the process mining literature.	36
2.3	A transactional database.	42
2.4	Frequent substructures of the database shown in Table 3.5.	42
2.5	A confusion matrix.	50
2.6	A contingency table for binary variables X and Y	55
2.7	Objective interestingness measures.	56
2.8	Common frequent subtree mining algorithms.	62
3.1	pre-order string encoding of T_{EX}	84
3.2	The <i>Flat Data Representation (FDT)</i> format of T_{EX}	87
3.3	The itemset format of T_{EX}	88
3.4	Mapping node labels to integers.	100
3.5	The <i>FDT</i> of T_{HOS}	102
3.6	The itemset format of T_{HOS}	102
4.1	Structural characteristics of data sets.	109
4.2	Average internal and external similarity for different clustering options.	110
4.3	A clustering result at $k = 4$	110
4.4	Prediction accuracy and coverage rates for each class.	115
5.1	The flat data representation of T_{cl}	129
5.2	The itemset format of T_{cl}	129
5.3	The <i>FDT</i> representation of T_{eval}	130
5.4	The itemset format of T_{eval}	130
5.5	Structural characteristics of the datasets used in Chapter 5.	134
5.6	The accuracy and coverage rates of <i>DSMC</i> when different rule strength types are applied to the <i>CSI-2</i> and <i>Hospital</i> datasets. Note that Th=Threshold.	139
5.7	The accuracy rates on ten-fold cross-validation of the <i>CSI-2</i> dataset.	140
5.8	Number of rules.	142
5.9	<i>DSMC+</i>	143
5.10	<i>XRules+</i>	144

5.11	Accuracy and coverage rates for individual classes in the <i>CSLOG</i> dataset.	147
5.12	Instances correctly(+)/incorrectly(-) classified by <i>DSMC</i> and <i>XRulesS</i> .	148
5.13	Number of rules of the hospital and <i>CRM</i> dataset.	150
5.14	Accuracy rates of the hospital dataset.	150
5.15	Coverage rates of the hospital dataset.	151
5.16	Accuracy rates of the <i>CRM</i> dataset.	151
5.17	Coverage rates of the <i>CRM</i> dataset.	151
5.18	A sequence of steps to predict an outcome of a running process instance.	153
5.19	Predicting whether running process instances comply to a linear temporal logic rule with a minimum support threshold of 1%–Hospital dataset.	154
5.20	Predicting whether running process instances comply to a linear temporal logic rule with a minimum support threshold of 5%–Hospital dataset.	154
5.21	Predicting whether running process instances comply to a linear temporal logic rule in (Maggi et al., 2014).	154
5.22	Predicting whether completed process instances comply to a linear temporal logic rule with a minimum support threshold of 1%–Hospital dataset.	154
5.23	Predicting whether completed process instances comply to a linear temporal logic rule with a minimum support threshold of 5%–Hospital dataset.	155
5.24	Predicting whether completed process instances comply to a linear temporal logic rule in (Maggi et al., 2014).	155
5.25	Predicting whether running process instances lead to a loan approval with a minimum support threshold of 5%– <i>DFI</i> dataset.	156
5.26	Predicting whether running process instances lead to a loan approval with a minimum support threshold of 10%– <i>DFI</i> dataset.	156
5.27	Predicting whether completed process instances get to a loan approval with a minimum support threshold of 5%– <i>DFI</i> dataset.	156
5.28	Predicting whether completed process instances get to a loan approval with a minimum support threshold of 10%– <i>DFI</i> dataset.	157
5.29	Predicting running process instances would be labelled as simple or complex with a minimum support threshold of 1%–Telephone Repair dataset.	157
5.30	Predicting running process instances would be labelled as simple or complex with a minimum support threshold of 5%–Telephone Repair dataset.	157

5.31	Predicting completed process instances would be labelled as simple or complex with a minimum support threshold of 1%—Telephone Repair dataset.	158
5.32	Predicting completed process instances would be labelled as simple or complex with a minimum support threshold of 5%—Telephone Repair dataset.	158
5.33	A sequence of steps to generate a recommendation model for a running process instance.	159
A.1	Accuracy rates for the <i>CS1-2</i> dataset	179
A.2	Accuracy rates for the <i>CS2-3</i> dataset	180
A.3	Accuracy rates for the <i>CS3-1</i> dataset	181
A.4	Coverage rates for the <i>CS1-2</i> dataset	182
A.5	Coverage rates for the <i>CS2-3</i> dataset	183
A.6	Coverage rates for the <i>CS3-1</i> dataset	184

Acronyms

BAM Business Activity Monitoring. 27

BI Business Intelligence. 27

BPI Business Process Intelligence. 27

BPM Business Process Management. v, xi, 1, 2, 6, 17, 19, 20, 33, 66

BPMN Business Process Model and Notation. 21

BPMS Business Process Management Systems. 1, 20, 21, 25, 26, 29, 39

CART Classification and Regression Tree. 48

CEP Complex Event Processing. 27

CRM Customer Relationship Management. 1

DFI Dutch Financial Institute. viii, xiii, 155, 161, 162

DSM Database Structure Model. vii, xi, xii, 73, 74, 78–86, 88–92, 94, 97, 100, 101, 104, 105, 130

EPC Event-Driven Process Chains. 21

EPLA Exploratory Process Log Analysis. 103

ERP Enterprise Resource Planning. 1, 20

FDT Flat Data Representation. xv, 85–88, 102, 110, 116, 117, 124, 128, 130

FPCT Flat Position-constrained Top-left Mirror. 84, 85, 88–91, 100

FPR False Positive Rate. 51, 52

ID3 Iterative Dichotomiser. 48

MXML Mining Extensible Markup Language. 6

PA Process Analysis. 27

PAIS Process-Aware Information Systems. 1, 4, 8

PCT Position-constrained Top-left Mirror. 83, 84, 88, 91

PI Process Intelligence. 27

TPR True Positive Rate. 50–52

WFM Workflow Management. 20

XES Extensible Event Stream. 6, 8, 9

XML Extensible Markup Language. 8

YAWL Yet Another Workflow Language. 21

Chapter 1

Introduction

According to Davenport (1993), a business process is a structured, measured set of activities whose purpose is to produce a specified output for a particular customer or market. There are many other definitions of business processes, but in general, they are “relationships between inputs and outputs, where inputs are transformed into outputs using a series of activities, which add value to the inputs” (Aguilar-Savén, 2004).

When business processes become more complex, automatic and systematic process management is required. *Business Process Management (BPM)* has gradually emerged as a solution to this need. *BPM* is defined as (van der Aalst, 2013):

methods, techniques, and tools to support the design, enactment, management, and analysis of operational business processes.

Before computer systems were invented, business processes were written in text and executed manually. Nowadays, most business processes are supported by means of *Business Process Management Systems (BPMS)*. Systems such as email clients, inventory software and account systems are able to assist in the execution of a business process. However, such systems are not built with a strong focus on business processes or a workflow engine. On the other hand, systems that have an explicit notion of process and are “aware” of business processes are called *Process-Aware Information Systems (PAIS)*. Some examples of these are *Enterprise Resource Planning (ERP)* systems, *Customer Relationship Management (CRM)* systems, *Websphere*, etc.

In *PAIS*, all process execution data such as *events*, *timestamps*, *resources* and *activities* are archived in *event logs* or *process logs*¹. This type of data can be analysed for process optimisation or improvement. For example, a university database contains information about students, activities (e.g. register, withdraw) and associated times, which can be considered as the resources, activities, and events of an event log, respectively. These data, together with the drop-out information, can be used to build a prediction model, and students who have a high probability of dropping out are flagged

¹the terms event log and process log are used interchangeably

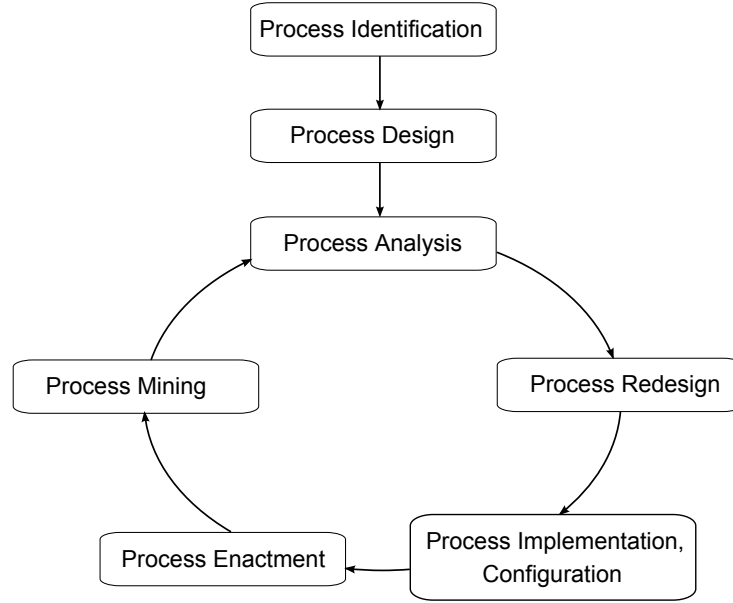


Figure 1.1: A *BPM* life cycle.

up to the university managers. The insights derived based on the analysis of the university process log clearly play an important role in the curriculum design and other decision making processes.

However, event logs are often very large in terms of the number of records and they are usually represented in a format that is not user-friendly. Hence, sophisticated, robust methods and tools are needed to extract hidden knowledge from event data to optimise or improve business processes. Many data mining techniques have been designed specifically for mining event logs; nevertheless, most of them focus on the discovery of process models or the use of available models as a starting point for further analysis. In this thesis, a direct mining approach for *XML*-based process logs is investigated.

This chapter starts with a description of *Business Process Management (BPM)*. Then, an overview of the process mining field is given in Section 2.2. Subsequently, the motivation of this thesis is described in Section 1.3. The main research questions and aims are discussed in Section 1.4. The contributions and impacts of our study to the data and process mining field are reported in Section 1.5. Lastly, the outline of this thesis is listed in Section 1.6.

1.1 Business Process Management

The *BPM* field is best viewed from the perspective of its life cycle. Fig. 1.1 shows a *BPM* life cycle consolidated from (Dumas et al., 2013; van der Aalst, 2011a; Weske, 2007). The seven phases of the life cycle are described as follows.

- In the process identification phase, all business processes in an organisation are identified and delimited.
- In the process design phase, the process analyst and domain experts cooperate to create as-is process models (see Section 2.1.2) that reflect the shared understanding of different stakeholders in an organisation. Note that process models are often described by graphical notations due to their intuitive structure. In addition, process models can be represented in formal languages, which enables automation.
- In the process analysis phase, business processes are analysed qualitatively and quantitatively. In qualitative methods, process analysts would interview various stakeholders to identify which part of the process has issues or needs improvements. A number of analysis methods could be performed such as value-added analysis, root cause analysis or impact assessment (Dumas et al., 2013). In quantitative methods, the process analyst would perform a process performance analysis, flow analysis, apply queue theory or conduct simulation (Dumas et al., 2013).
- In the process redesign phase, to-be process models are mainly created from the modification of as-is process models, which is a direct result of the analysis in the previous phase.
- In the process implementation and configuration phase, organisational changes are made and the to-be process models are automated by computer programs. If the software is already up and running, only process parameters are to be configured.
- In the process enactment phase, business processes are executed and event data are logged.
- In the process monitoring and controlling phase, the main task is to analyse on-line (monitoring) or offline (controlling) event data. In both cases, notifications are reported to decision makers when bottlenecks or deviations are detected. If the problems are not severe, e.g. an activity requires more resources at peak periods, only process reconfigurations are needed. The process performances are evaluated against predefined objectives, and served as inputs to the process analysis phase.

Unlike other phases, process controlling and monitoring have only attracted significant attention in recent years, due to the advancement of processing and storage technologies. The rapidly-increasing speed of computing hardware enables more applications of data mining (see Section 2.3 for more details) to process monitoring. In

addition, large-capacity storage systems are becoming cheaper, leading to more event data being archived for analysis. The seminal works of Cook and Wolf (1995, 1999) and van der Aalst et al. (2004) have laid the foundations for the process mining field, which is considered to encompass process monitoring, controlling and other evidence-based process analysis techniques.

1.2 Process Mining

Event logs are generated by information systems and are the starting point of process mining (van der Aalst, 2011a). With an event log at hand, a variety of data analysis methods can be used to gain insights into how a business process is executed and to identify areas that can be improved. It is often assumed that an event log contains instances of a single business process. Each instance in an event log is called a *process instance*. Each process instance, also known as a *case*, is comprised of a set of events. A *trace* is a case whose events are ordered sequentially according to the time when they occur.

Fig. 1.2 shows an architecture of process mining, adjusted from (van der Aalst, 2011a). In *PAISs*, event logs are comprised of current data (data about processes that are running) and past data (data about processes that are already complete). As-is process models are descriptive, whereas to-be process models are normative, which means that the execution of business processes is constrained by these models. There are four categories of tasks in process mining, i.e. cartography, navigation, auditing, and root cause analysis.

In the cartography category, the three main tasks are process *discovery*, process *diagnosis*, and process *enhancement*. The first task involves the identification of a (as-is) process model that is “representative” for of the behaviour seen in the event log (van der Aalst, 2011a). The second task deals with searching for logical inconsistencies in process models, or analysing process performance indicators such as time, cost, or quality (using techniques like simulation, integer programming, etc.). The third task enhances the discovered process models by adding different perspectives to them e.g., organisational, social network perspective, using a variety of data attributes in event logs.

In the auditing category, the main tasks are to *compare*, *check*, and *promote*. The first task is to compare to-be process models with as-is process models to identify the deviation of reality from what was planned. The second task is about checking the compliance of an event log to its corresponding to-be process model. Based on the results of the first task, it is possible to promote parts of the as-is model to a new to-be model (van der Aalst, 2011a).

In the navigation category, some key tasks are to *explore*, *predict*, and *recommend*.

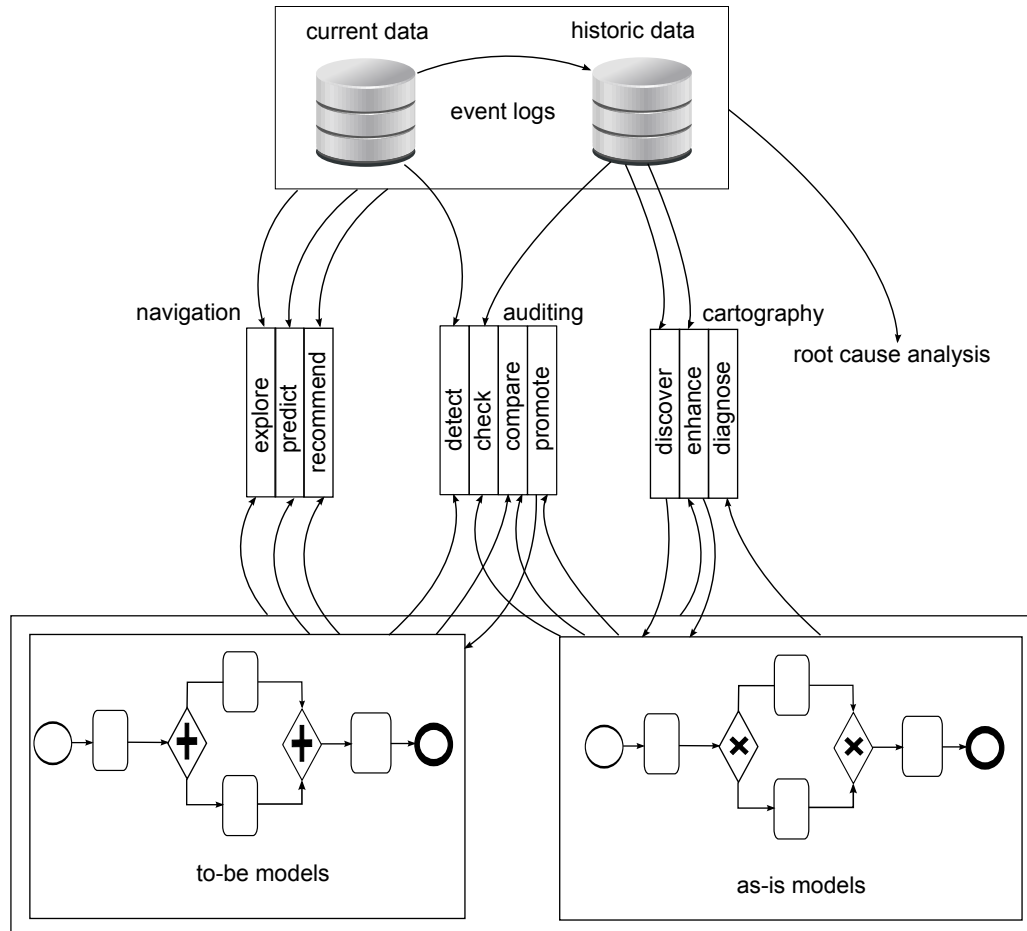


Figure 1.2: A process mining architecture.

In exploration tasks, business processes are visualised at run-time such that event data are displayed in accordance with the process models. In prediction tasks (van der Aalst, 2011a; Rogge-Solti and Weske, 2013; Pika et al., 2013; van der Aalst et al., 2011), the purpose is to measure how likely an event is to occur out of the possible outcomes in a running process instance. Possible outcomes of a process instance can be remaining processing time (flow time), cost of a process execution, whether a process instance is successful or failed, whether a business goal is achieved or not, whether a performance threshold is reached or not, which resource is assigned to an activity, or whether an undesirable outcome happens or not. A recommendation task (van der Aalst, 2011a; Conforti et al., 2013; Nakatumba et al., 2012; Maggi et al., 2014) is to suggest what to do next in order to obtain a desirable outcome (e.g. minimising cost, resource usage, or remaining processing time) given a partial trace of a running process. A recommendation might be the next activity to be executed, or selecting a resource for the current task.

In root cause analysis (or fault diagnosis), contribution factors to an undesirable outcome of a business process, such as overtime fault, are identified (Suriadi et al., 2013).

1.3 Motivation

From the process mining architecture described in Fig. 1.2, it can be seen that most process mining tasks, e.g. model enhancement, comparison of as-is and to-be models, require the availability of process models. In the literature, process model discovery has been the focus of a majority of studies in process mining. Generally, process models are an effective means of describing processes; therefore, their presence is seen in all phases of *BPM*. However, the availability of a correct and up-to-date process model in a constantly changing business environment is not always certain. Furthermore, process models are sometimes hard to obtain, especially in complex processes.

With the advent of *Mining Extensible Markup Language (MXML)* and *Extensible Event Stream (XES)* standards, the two *XML*-based² representations of event data, an interesting question arises; “whether can *XML* data mining techniques be used to directly discover interesting knowledge patterns from *MXML/XES* data without the requirement of a process model?”. This question will be addressed in the remainder of this section. An overview of process mining methods that are not based on process models is provided in Section 1.3.1. *XML*-based event logs are described in Section 1.3.2. Finally, some tentative answers to the above question are presented in Section 1.3.3.

1.3.1 Process Mining without Process Models

Event data have been used not only in the process mining field but also in other areas, such as debugging, fault management, intrusion detection and prevention (Makanju et al., 2012); network logs are used for network monitoring and network administration (Vaarandi, 2004), and equipment logs are used for predictive maintenance (for the purpose of reducing equipment downtime) (Sipos et al., 2014). It is worth noting that in such types of event data, the logs are often analysed without using process models.

As mentioned earlier, most process log analysis methods consider process models as a-priori (conformance checking and model enhancement) or a-posteriori knowledge (process discovery) (for more details, see Section 2.2). Recently, a few studies on the direct mining of process logs have been conducted, mostly related to the *prediction*, *recommendation*, *root cause analysis*, outlier detection, and association rule mining tasks. These tasks are presented in turn as follows.

In traditional process mining, the prediction task is about using statistical analysis over simulation/replay of process instances on the process models. Recently, there have been a few attempts at applying traditional data mining techniques directly to process logs, such as in the work of Maggi et al. (2014). In this study, the class la-

²<http://www.w3.org/XML/>

bels are outcomes or performance indicators of a business process. Feature vectors are selected from the training data, mainly from cases or events' attributes. For example, a case attribute such as *patient age* might be a good feature in predicting the outcome of a treatment (e.g. duration of the treatment). In another example of an insurance company, an event attribute of the process *make a claim*, e.g. *amount of money requested*, might be a useful feature in predicting fraudulent claims. An abnormal sequence of events could signal a fraud, such as skipping an important verification step and approving claims that normally ought to be rejected.

In recommendation tasks, it is also possible to apply data mining techniques directly to event data such as the works in (Conforti et al., 2013) and (Maggi et al., 2014). A common approach is to first build a prediction model, e.g. decision tree, or regression model. Note that training instances are labelled by a performance indicator or a specific goal. Then, the partial traces (a subset of a trace that includes current and previous activities) are appended with each possible activity or resource, resulting in a set of extended traces. The third step involves matching the extended traces with the condition parts of classification rules. The probability of reaching a specific goal from each extended trace is determined by the rules that are triggered and their rule strengths, which can be measured by different methods, e.g. confidence, lift. Essentially, the recommendation task only differs from the prediction task in the selection of class labels and the application of the prediction model. It is worth noting that process models could play a supplementary role in the process, i.e. identifying the next possible activities or resources from the current event.

In root cause analysis, classification methods are used to identify attributes that cause an undesirable outcome of a business process. The process and event's attributes can be used directly, or may have to be aggregated/derived from other attributes (Suriadi et al., 2013). For example, the cycle time of a process could be used as a feature when building a prediction model; however, in many cases, this data is not given in the process's attributes, but is derived from the timestamps of the first and last events of the process.

Outlier detection in event logs has just begun to receive attention in recent years. Its objective is to identify process instances whose behaviours deviate from those of most others individuals in the process logs. A process instance conforming to the specification of a process model can still be an outlier (Ghionna et al., 2008); hence, process models are not necessarily a requirement for the outlier detection task. Instead, clustering techniques are used to split a process log into groups of similar process instances and outlying instances belong to groups that have a significant distance to all other groups are individuals that have large similarity distance to all other groups.

Despite its numerous applications in a wide variety of fields, association rule discovery has been used in only a handful of works in process mining such as (Rebuge

and Ferreira, 2012; Yang and Hwang, 2006; Swinnen et al., 2012). Association rule discovery can be utilised in the event data of an automated *pay-by-phone* system. For example, a company might be interested in finding out the reasons that have led to clients abandoning calls. Although simulation is often performed to get an overall feel of how the system is running, associative rule discovery may provide insights into specific aspects, e.g. discovering associations between events/customer characteristics and abandoned calls. Using various filters on the data/rules and then extracting associative rules can provide focused answers quickly.

In summary, although most process mining tasks have been based on process models, the number of methods that apply data mining methods directly to process logs has been on the rise. An advantage of this approach is that the focus has shifted from the control-flow aspect of a business process to the whole data, thereby alleviating the dependency on process models.

1.3.2 The Advent of *XML*-based Event Logs

In *PAISs*, there has been no specification of how event data are logged. This hinders the communications between information systems and process mining tools. Important data attributes needed for process mining might be missing, or the data might have to go through many pre-processing steps to be used effectively. There should be standardisation regarding which information should be recorded, and how the data are organized in an event log.

Extensible Markup Language (XML)-based languages are the best candidates for the representation of event logs due to their simplicity, portability, self-description and extensibility. In addition, it is easy to convert data from any source into *XML* (Suciu, 2000) documents, which are then made available to *XML* data mining applications. *MXML* (van Dongen and van der Aalst, 2005), a variant of *XML*, was proposed as a representation language for event logs in 2005 and quickly gained acceptance from the process mining community (Verbeek et al., 2011). *MXML* uses standardised tag names to represent core elements of an event log. For example, process instance, event, transition, and timestamp are labelled as `<ProcessInstance>`, `<AuditTrailEntry>`, `<EventType>`, and `<Timestamp>` in event logs, respectively.

Despite its effectiveness, the meta model of *MXML* is restrictive and non-extensible. An *XML*-based language, known as *XES* (Verbeek et al., 2011), was created as an enhancement over *MXML*. In *XES*, some standardised tags of *MXML* are renamed; e.g. `<AuditTrailEntry>` was changed to `<event>`, or replaced, e.g. `<EventType>` has been replaced by attributes `key="lifecycle:transition"`. In addition, the semantics of attributes are defined in extensions, which are comprised of related attributes, along with their data types and meanings. For example, the time extensions

defines the attribute *timestamp* with its associated data type *string* and definition “the date and time, at which the event occurred”. It is also possible to introduce user-defined extensions. Furthermore, *XES* allows us to assign an identity to each event through *event classifiers*.

A motivating example of an *XML*-based event log was created, based on a real-life event log in the hospital domain (van Dongen, 2011; Bose and Aalst, 2012). The log file, which is in *XES* format, is presented in Listing 1.1. The extensions definition and other details are omitted to simplify the example. Listing 1.1 shows part of an artificial event log representing a treatment process in a hospital. The data inside each element `<trace>` represents a single process instance. The attribute `<concept:name>` describes an activity of the patient, the attribute `<org:group>` represents the location where the activity occurs, and the attribute `<AgeGroup>` represents the age of the patient.

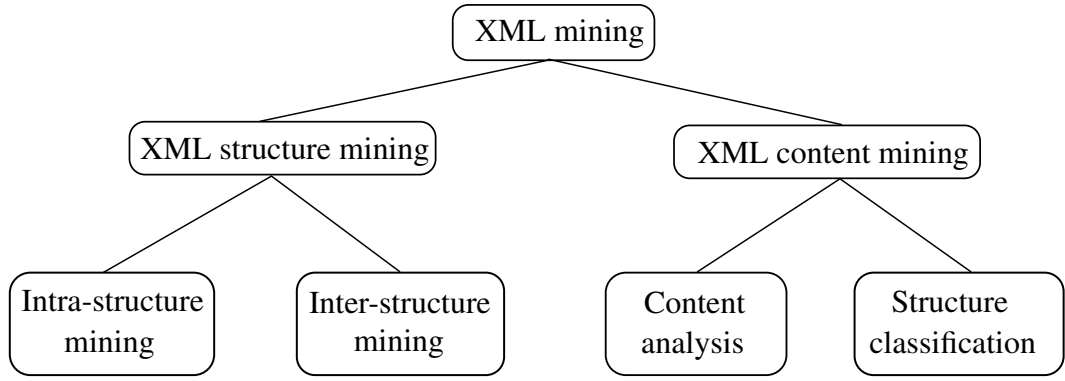
Listing 1.1: A part of an artificial event log in *XES* format representing a treatment process in a hospital

```
<trace> <string key="AgeGroup" value="21-65"/>
  <event> <string key="concept:name" value="Consultation"/>
    <string key="org:group" value="RadioTherapy"/> </event>
  <event> <string key="concept:name" value="Administration"/>
    <string key="org:group" value="RadioTherapy"/> </event>
  <event> <string key="concept:name" value="BloodTest"/>
    <string key="org:group" value="GeneralLab"/> </event>
</trace>
<trace> <string key="AgeGroup" value=">65"/>
  <event> <string key="concept:name" value="Consultation"/>
    <string key="org:group" value="ObstetricsClinic"/> </event>
  <event> <string key="concept:name" value="Administration"/>
    <string key="org:group" value="ObstetricsClinic"/> </event>
  <event> <string key="concept:name" value="CytologicEvaluation"/>
    <string key="org:group" value="Pathology"/> </event>
</trace>
<trace> <string key="AgeGroup" value="21-65"/>
  <event> <string key="concept:name" value="Consultation"/>
    <string key="org:group" value="RadioTherapy"/> </event>
  <event> <string key="concept:name" value="Administration"/>
    <string key="org:group" value="RadioTherapy"/> </event>
  <event> <string key="concept:name" value="CytologicEvaluation"/>
    <string key="org:group" value="Pathology"/> </event>
</trace>
<trace> <string key="AgeGroup" value="<21"/>
  <event> <string key="concept:name" value="Consultation"/>
    <string key="org:group" value="ObstetricsClinic"/> </event>
  <event> <string key="concept:name" value="PhoneConsultation"/>
```

```
<string key="org:group" value="ObstetricsClinic"/> </event>
<event> <string key="concept:name" value="Administration"/>
  <string key="org:group" value="ObstetricsClinic"/> </event>
</trace>
<trace> <string key="AgeGroup" value="21-65"/>
  <event> <string key="concept:name" value="CTAbdomen"/>
    <string key="org:group" value="Radiology"/> </event>
  <event> <string key="concept:name" value="MRIAbdomen"/>
    <string key="org:group" value="Radiology"/> </event>
  <event> <string key="concept:name" value="Microscopic Examination"/>
    <string key="org:group" value="Microbiology"/> </event>
</trace>
<trace> <string key="AgeGroup" value=">65"/>
  <event> <string key="concept:name" value="CTAbdomen"/>
    <string key="org:group" value="Radiology"/> </event>
  <event> <string key="concept:name" value="MRIAbdomen"/>
    <string key="org:group" value="Radiology"/> </event>
  <event> <string key="concept:name" value="FlamePhotometer"/>
    <string key="org:group" value="GeneralLab"/> </event>
</trace>
```

The fact that *MXML* and *XES* became the standards of event logging opens up opportunity for *XML* data mining methods (Feng et al., 2003; Nayak et al., 2002; Tagarelli, 2012) to be applied to this data. Two main approaches on *XML* data mining are mining structures alone and mining both structures and content (Nayak et al., 2002) (see Fig. 1.3). This thesis focuses on the former approach—the work in (Piernik et al., 2015) shows that structural mining on *XML* data is scalable in big and complex datasets while the other is not. In the structural mining approach, *XML* data can be represented in many forms such as graphs, trees, path sets, tag vectors and time series. In those, tag vectors are suitable for light weight documents, while trees and paths are appropriate for documents that are homogeneous (sharing the same tag vocabulary) (Piernik et al., 2015). Since event logs that comply to the *MXML* or *XES* standard have repetitive substructures which include nodes ‘event’ and their attributes, tree representation is a plausible choice. Furthermore, among different types of tree that can be used to represent event logs, this work focuses on rooted labelled ordered trees (Zaki, 2002; Hadzic, 2012) because the type of event and the order of when events occur are important in process log analysis. Data mining methods performed on a set of trees (forest) (Chi et al., 2005b, 2004a), correspond to the *inter-structure mining* according to the taxonomy shown in Fig. 1.3 (Nayak, 2008).

The study in (Feng et al., 2003) emphasised the importance of tree-structured patterns in that they are “more natural, informative and understandable”. Moreover, subtree patterns can be distinguished by their context positions. For example, Fig. 1.4(a) and Fig. 1.4(b) shows two subtree patterns representing movies that have the same di-

Figure 1.3: An *XML* mining taxonomy (Nayak, 2008).

rector and actor; however, the context of each pattern is different because one belongs to the category of ‘Thriller’ and the other belongs to ‘Comedy’.

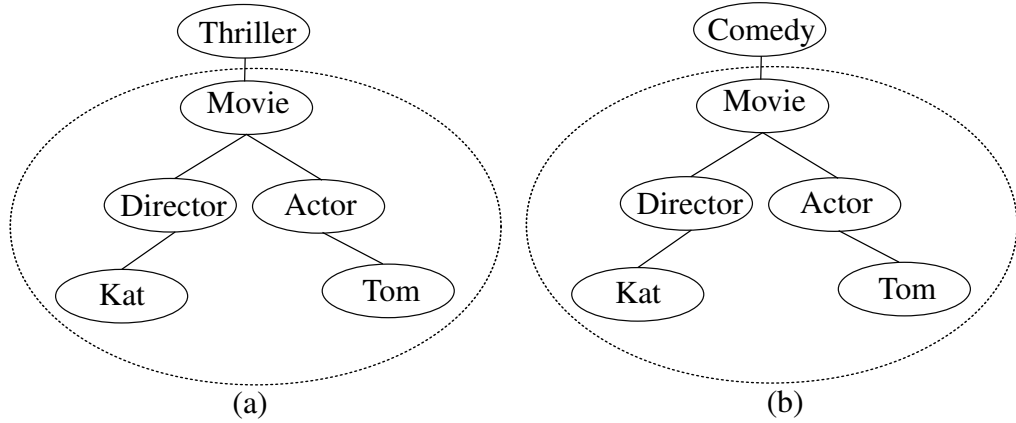
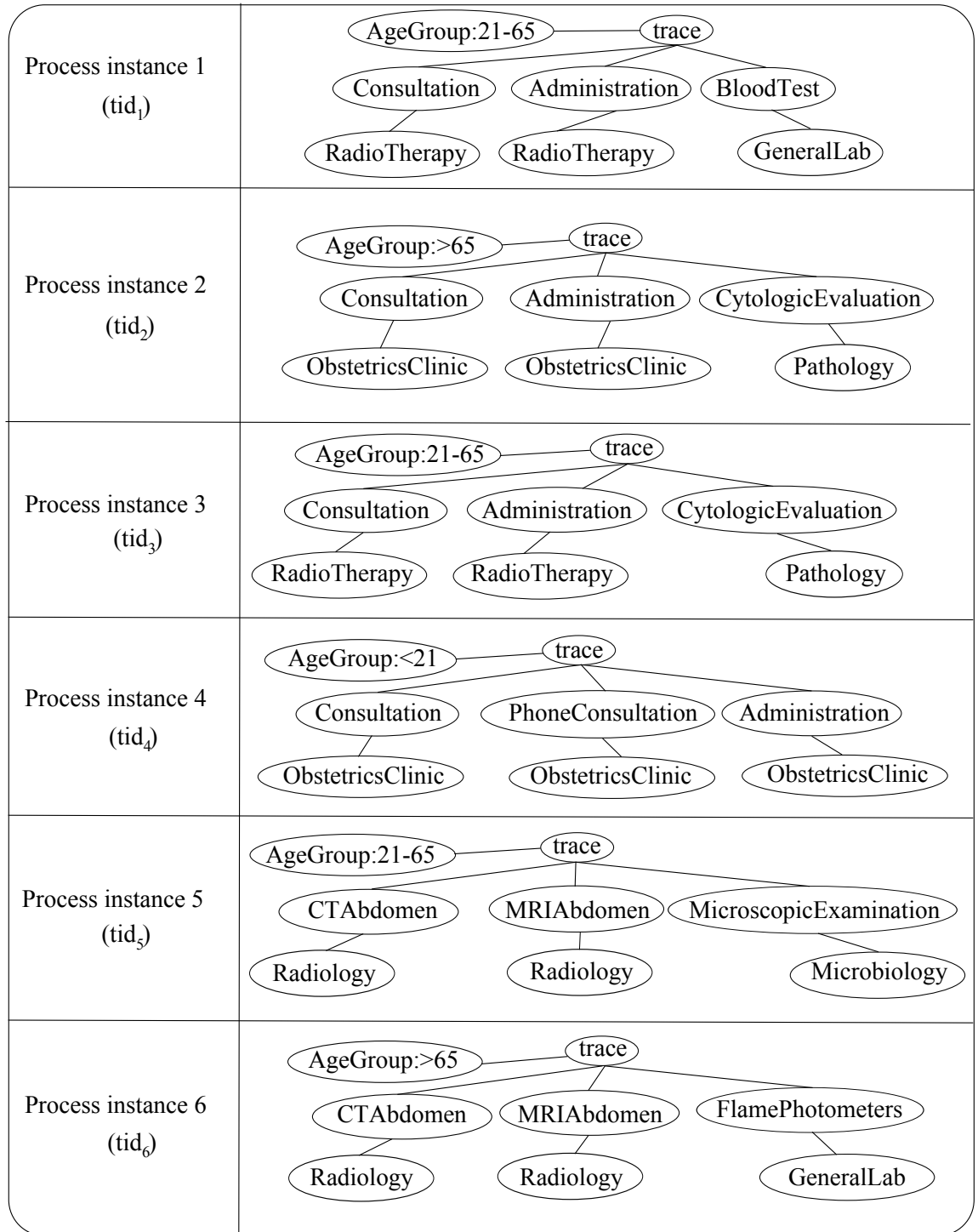


Figure 1.4: A subtree pattern in different contexts.

To utilise tree-structured mining methods, the event log described in Listing 1.1 needs to be converted into a rooted, labelled, ordered tree database, called T_{HOS} , which is shown in Fig. 1.5. Each node in a tree represents an element/attribute in the *XES* document. Note that the keys of the attributes are removed to make the figure less cluttered. Additionally, the relationship between nodes representing *org:group* and nodes representing *concept:name* is changed from sibling to parent-child for simplicity reasons. The next section provides an overview of how *XML* mining methods are applied to tree-structured data and examples are given on T_{HOS} .

1.3.3 Examples of *XML* Mining Methods and Their Applications

Frequent pattern mining (Han et al., 2007a; Chi et al., 2004a; Zaki, 2002; Hadzic et al., 2011b) is to identify frequent structures in a database. Applying a frequent pattern mining algorithm to *MXML/XES* event logs would return groups of activities and/or attributes that frequently occur together. The frequency of the occurrence is called *support*. For example, applying the *TreeMiner* algorithm (Zaki, 2005b) to the tree

Figure 1.5: Tree database T_{HOS} represents the event log shown in Fig. 1.1.

database T_{HOS} with a minimum support of two would return many frequent subtrees (defined in Section 2.4.3). Two of them are presented in Fig. 1.6. The frequent subtree in Fig. 1.6(a) illustrates that there are at least two patients aged between 21 and 65 following a sequence of activities at some stages in their treatment process: *Consultation*, then *Administration*, both at the *RadioTherapy* department. The frequent subtree in Fig. 1.6(b) illustrates the same sequence of activities, but the locations at which the activities took place are different.

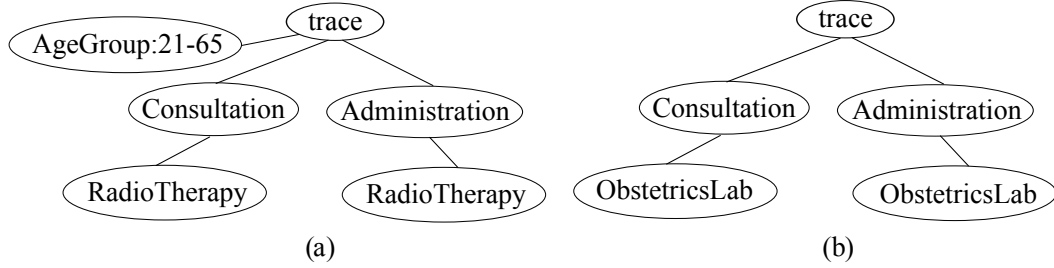


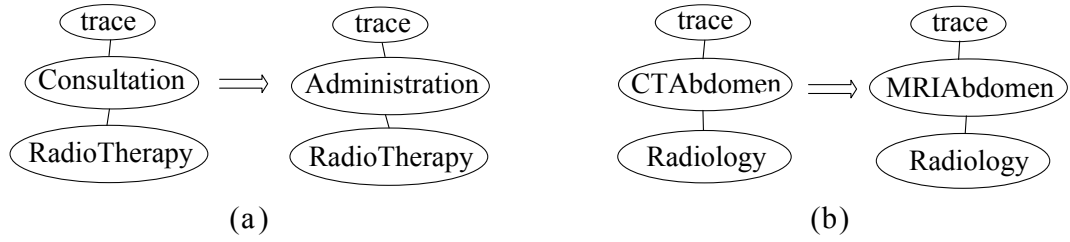
Figure 1.6: Two frequent subtrees found from T_{HOS} .

Association Rule Mining

Association rule mining (Agrawal et al., 1993), a well-researched area in data mining, identifies rules in the form of $X \Rightarrow Y$, where X and Y are disjoint sets of items. An association rule expresses the probability of a transaction containing X also containing Y .

Association rule mining has been used to identify business rules from repositories of business process designs, and was then used as a prior knowledge for further analysing, verifying, and modifying process designs (Polpinij et al., 2010). In another study, the χ^2 test was performed on an insurance company's database to identify the correlation between two suspected characteristics of process instances (Caron et al., 2013).

Using association rule mining on *XML*-based event logs would uncover interesting relationships between activities, resources, and other properties. For tree-structured databases, the antecedent and consequent of a rule are subtrees. From the databases shown in Fig. 1.5, many tree-based association rules can be discovered, two of which are shown in Fig. 1.7 as an example. The association rule in Fig. 1.7(a) illustrates that for any patient treatment process instance, if there is a *Consultation* activity at the department of *RadioTherapy*, an *Administration* activity in the same department would also be likely to happen. The application of association rule mining to *XML* documents has been studied in (Mazuran et al., 2009a; Moradi and Keyvanpour, 2015; Feng and Dillon, 2005).

Figure 1.7: Two association rules found from T_{HOS} .

Classification

The classification task is to construct a prediction model based on training instances, whose class labels are known, and using this model to predict the class label of test instances. In the context of process mining, each process instance in the training data should be labelled according to a business goal. A prediction model is then learned from the training data so that it is possible to predict outcomes of ongoing process instances.

As an example, the process instances shown in Fig. 1.5 are used as training data. The first four process instances are labelled with class *high risk* and the last two instances are labelled with class *low risk*. Fig. 1.8(a) shows an example of a prediction model which includes one rule. The subtree on the left (rule antecedent) represents the condition of the rule and the value on the right (rule consequent) represents the predicting class. If a test process instance satisfies the condition, i.e. the subtree in the antecedent occurs in the test instance (see Section 2.4.2 for the definition of tree occurrence), then it is classified as the predicting class in the rule consequent, along with a confidence value. The test instance in Fig. 1.8(b) satisfies the condition in Fig. 1.8a. Hence, it is classified as “low risk” with a confidence value of $x\%$.

XML classification based on structure alone was investigated in (De Knijf, 2007; Costa et al., 2013b; Garboni et al., 2006; Zaki and Aggarwal, 2006; Costa et al., 2011).

Clustering

The clustering task (Jain et al., 1999) is to identify groups of instances that share some commonalities. When applying *XML*-based or tree-structured clustering methods (Algergawy et al., 2011; Dalamagas et al., 2005; Aggarwal et al., 2007; Costa et al., 2013a; Nayak, 2008) to *MXML/XES* event logs, it is possible to identify multiple variants of a process model, or to separate process instances that belong to different process models, but were mixed into one event log. For example, applying a tree-based clustering algorithm to the database shown in Fig. 1.5 would return two clusters of process instances; the first cluster includes the first four process instances which are mainly involved with consultations and administration activities, and the second cluster contains the last two process instances which are mainly involved with examination activities.

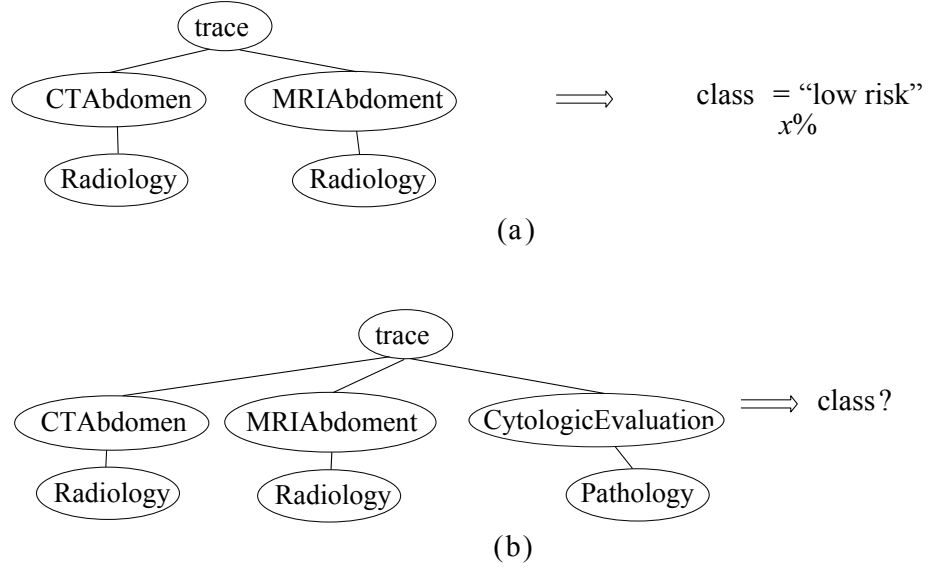


Figure 1.8: (a) A rule-based prediction model learned from the tree database in Fig. 1.5. (b) A test process instance.

1.4 Research Questions and Objectives

Current process mining techniques focus on process discovery, which is to identify a control-flow of a business process by applying machine learning or data mining methods on event logs. Without process models, common process mining tasks such as conformance checking and process enhancement (van der Aalst, 2011a) would be impossible. However, it is hard to discover process model from event logs due to noise in data or unstructured processes (also called spaghetti-like processes) (van der Aalst, 2011a). Additionally, even though event logs are archived in *XML*-based languages, but are then extracted and converted to a suitable format decided upon by a specific process mining algorithm. This adds unnecessary complexities to process mining. To the best of our knowledge, there are no methods that directly mine process logs in *XML* format. This gap helps us identify our research question, which can be summarised as “In what way can *XML* data mining techniques be applied to event logs to discover hidden knowledge without using process models?”

To answer the above research question, the following tasks should be done.

- Analyse the strengths and weaknesses of *XML* data mining methods for process mining tasks.
- Devise an integrated approach for analysing an *XML*-based process log without using process models.

This thesis proposes the use of frequent subtree mining techniques (Zaki, 2002) and position-constrained frequent subtree mining method (Hadzic, 2012) on solving

process mining tasks. It is known that there is no “one-size-fit-all” technique, thus the thesis is expected to provide the improvements to the existing techniques.

1.5 Research Contributions and Significance

Research Contributions

This thesis introduced an integrated method, called *PCFSM*, which is based on the position-constrained frequent subtree mining method to directly mine *XML*-based event logs. Although this method requires the conversion of the tree database to a flat data representation, the resulted format enables a wide variety of traditional data mining techniques to be (indirectly) applied upon. Moreover, it was shown that no data loss after the conversion process. In addition to the *PCFSM* method, an exploratory process log analysis method, which is capable of analysing a complex process log in an unbiased way, was proposed. A set of synthetic and real-world datasets were used to evaluate the proposed methods. In another experiment on an event log, the time performance of the position-constrained frequent subtree mining was compared to that of other state-of-the-art frequent subtree mining methods such as *CMTreeMiner* (Chi et al., 2005a) and *DryadeParent* (Termier et al., 2008).

Another contribution of the thesis is the proposal of an associative classification method, called *DSMC*, which is based on the positioned-constrained frequent/closed subtree mining. This classification method is extensively evaluated on different types of datasets and compared with an associative classification method which is based on traditional frequent/closed subtree mining. Furthermore, two methods to improve *DSMC*'s classification accuracy were also suggested.

The capability of the *PCFSM* and *DSMC* methods were further exhibited through their applications to the prediction and recommendation task. With the availability of *DSMC*, it is possible to predict whether a running process instance would result in a successful execution according to a predefined business goal. Furthermore, the extension of the *PCFSM* method helps process owners to identify the activities that might increase the chance of obtaining a desired outcome.

Research Significance

Traditional process mining methods often require a process model, but in many cases, the process models are not available or are hard to obtain. This thesis offers a direct mining method on *XML*-based process logs, without the need for a process model. In addition, in this method, the semi-structured event logs are treated as a whole, which is in contrast to the traditional methods in which only relevant data attributes are extracted

and processed. This method is therefore easier to use and offers an alternative solution to traditional process mining methods.

By converting the *XML*-based event logs into a flat representation, classical data mining methods such as association rule mining, classification and clustering can be applied to event logs. The process analysts would benefit from using this method because classical methods are often simpler and more flexible than the proprietary methods. Note that the conversion of semi-structured data into flat format does not result in any loss of information as shown in (Hadzic, 2012; Hadzic et al., 2015). Furthermore, the flat data can be easily converted back to *XML* format (Hadzic, 2012).

The proposed process log analysis method is based on a structure-preserving tree mining approach, which makes it possible to locate frequently-occurring activities or attributes in process instances, whereas the discovered patterns using traditional frequent subtree methods are less informative. In addition, the patterns generated by our method can also be a set of disconnected subtrees which might be useful in some applications. Furthermore, by constraining nodes with their position, the integrated method is able to run at a much lower minimum support even in large and complex datasets, while the number of patterns does not increase as fast as other methods’.

1.6 Thesis Outline

This thesis is divided into six chapters as follows:

- Chapter 1 introduces the context of the thesis. An overview of the *BPM* and process mining field is then provided. An analysis of some process mining tasks and an introduction of *XML*-based process logs are given as the motivation for the study. After that, the chapter presents several research questions and objectives, and concludes with research contributions and significance.
- Chapter 2 provides a comprehensive background for this thesis. Important concepts of business process management and process mining are described. Data mining methods, formulas and concepts that are used throughout the thesis are presented next. Lastly, tree-structured data and *XML* mining methods are discussed.
- Chapter 3 starts with a motivating example that emphasises the need for a direct mining approach to process logs. Next, two tree-structured data mining approaches are presented. The structure-preserving tree mining approach is selected and described. After that, a position-constrained tree-structured process log analysis method, *PCFSM*, is proposed. In addition, an exploratory process log analysis method is recommended for process analysts who are not familiar with the data in an unbiased manner.

- Chapter 4 presents an evaluation of the *PCFSM* and the exploratory process log analysis methods. The proposed methods are tested on a number of synthetic and real-word datasets.
- Chapter 5 describes an associative classifier which is based on the position-constrained tree mining method. This classifier, called *DSMC*, is built as an extension of the *PCFSM* method. Heterogeneous and homogeneous datasets are used to evaluate the classification accuracy and coverage rates of the method. The *DSMC* was extended to predict outcomes of running process instances. Next, the *PCFSM* was extended to recommend suitable actions for users to prevent unwanted outcomes. Eventually, several techniques are suggested for the enhancement of the *DSMC* classifier.
- Chapter 6 concludes and outlines some possible extensions of the thesis.

Chapter 2

Literature Review

This chapter is dedicated to providing a solid background that subsequent chapters are built upon. In Section 2.1, an overview of the business process management is provided. Section 2.2 illustrates main concepts of process mining, which is the main theme of this thesis. To provide the readers common tools that can be used for process log analysis, Section 2.3 presents related data mining concepts. Section 2.4 discusses *XML* data mining methods that can be used for process log analysis. After that, structural *XML* classification is presented in Section 2.5. Section 2.6 recapitulates this chapter and lists the research gaps that needs to be addressed.

2.1 Business Process Management

This section provides the background of the business process management field. Firstly, the development of the notion *business process* over time is described. Secondly, the notion of *process model* is introduced. Each phase in the *BPM* life cycle is then discussed in subsequent sections.

2.1.1 History of *BPM*

In prehistoric times, things that served basic needs of human, e.g. fire, can be made individually, with little or no cooperation. The process of making products was often simple and can be performed in a few steps.

In medieval times, more sophisticated products were invented and their production process required trained people with specialised skills, e.g. shoes are made by shoemakers.

The industrial age introduced the concept of labour division, in which workers have to repeatedly perform one specific type of activity, which is one part of the whole production process. Due to the high complexity of the process, a managerial role was needed to coordinate activities and resources.

When making a product required a large number of people, different functional units such as purchasing, production, marketing, etc., were created and each of them is specialised in one aspect of the production process. These units had to work together, aiming at a single goal of making the product. However, at a certain level of complexity, it became extremely hard to manage a process that spans multiple departments and external entities without a rigorous management framework.

BPM is a body of methods, techniques, and tools that support the design enactment, management, and analysis of operational business processes (Ordys et al., 2007). Business managers are attracted to *BPM* because of its demonstrated ability to deliver improvements in organisational performance, regulatory compliance and service quality (Dumas et al., 2013). Other disciplines that are related to *BPM* includes Total Quality Management, Operations Management, Lean, and Six Sigma. These disciplines focus on different aspects of business process management and are not as encompassing as *BPM*.

In recent decades, information technologies began to take root in organisations and became the main driver of process innovations. An *ERP* system comprises of a set of integrated software modules that share a central database and each module is responsible for a functional unit of an organisation. *Workflow Management (WFM)* are built upon business process models (see next section) and its main function is to dispatch tasks to *process participants* (human actors who perform the activities of a business process). Over time, *ERP* systems and *WFM* systems were integrated each other and a larger system became known as *BPMS*.

2.1.2 Process Model

Business processes are often complex and involved with many process participants and interdependent activities. Hence, to help human and machine to comprehend business processes, a concise and formal description of them is needed. The process model of a business process is the description of that process. Process models can be represented in text, however, formal diagrams are preferred because they are concise and leave less room for misunderstandings. Process models are seen in all phases of a *BPM* life cycle.

Two essential elements of a process model diagram are activity and control nodes. An activity node describes a work that is performed by a resource (a process participant or a computer program). A control node illustrates the flow of execution between activities. There are many types of control-flow nodes such as parallel split, sequence, synchronisation, multi-choice, multi-merge, etc. (van der Aalst et al., 2003). In Fig. 2.1, a process model of a business process “serving food” is represented in *Petri net* notation. In this notation, an activity node corresponds to a *transition* which

is represented by a square. The transitions in the figure are “receive order”, “make entrée”, “make main course”, “make dessert”, “serve entrée”, “serve main course”, “serve dessert”. The transitions are connected through places, which are circles in the diagram and responsible for routing the activity flow. The three places after “receive order” acts like a parallel split that allows “make entrée”, “make main course”, and “make dessert” to occur independently.

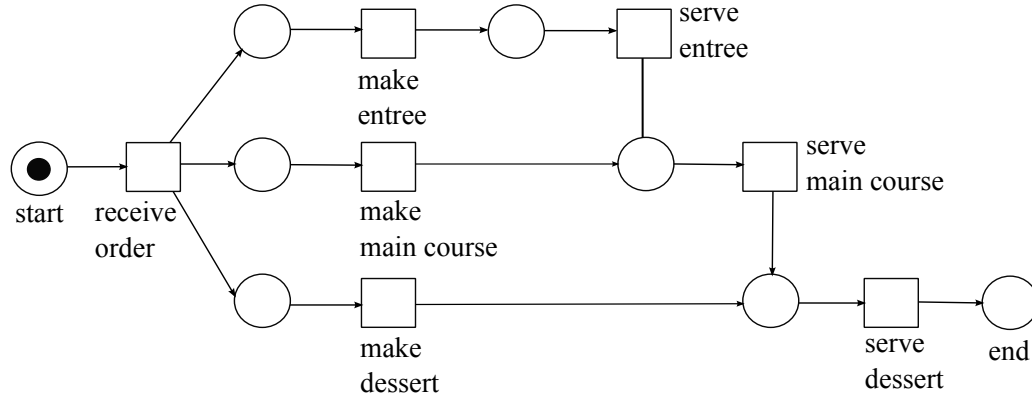


Figure 2.1: An example of process model in Petri net notation.

There are two main types of process models, i.e. as-is and to-be process models. As-is process models are observed or extracted from the current executions of a business process. They are descriptive, which means that they reflect how business processes are carried out in reality. In contrast, to-be process models are agreed among different stakeholders in one or many organisations and then used as a basis for the design, implementation and enactment of *BPMS*. To-be process models are normative, which means that they act like an instruction manual for resources to follow upon.

Process models can be viewed not only from a control-flow perspective but also a data-flow perspective. In this perspective, information artefacts (Nigam and Caswell, 2003) are shown as an output or input of activity nodes, together with control nodes. For example, activity *Buy a ticket* requires information artefact such as *credit card number*.

A business process can be viewed from a number of perspectives. In an organisational perspective, process models are represented in social-network-like diagrams, depicting the handover of works among process participants. In a time perspective, timestamps and durations are annotated to activity nodes in process models. Each perspective offers a specific type of view on a business process model.

Many notations have been design for modelling business processes, e.g. Petri nets (Peterson, 1981), *Business Process Model and Notation (BPMN)* (Weske, 2007), *Yet Another Workflow Language (YAWL)* (ter Hofstede et al., 2010), and *Event-Driven Process Chains (EPC)* (Scheer, 1994).

2.1.3 Business Process Identification

The process identification phase includes two main activities, i.e. identifying main business processes and their boundaries in an organisation, and deciding the priority of each process, which is based on a number of criteria, such as profitability or customer satisfaction.

2.1.4 Process Design

In process design, process analysts have to work with domain experts to construct as-is process models. Process analysts know much about modelling but do not know how a business process works. On the contrary, domain experts know a lot about operational activities but do not have the modelling expertise.

Since designing a process model is not straightforward, a team including a mix of domain experts and process analysts are required in the modelling task. Process analysts are involved because most domain experts think in terms of a concrete case but not in an abstract way of how a business process is operated. In addition, domain experts are not familiar with process modelling notations, process analysts are needed to interpret models to natural language description.

Process models are built from data that can be extracted using the following methods:

- Evidence-based: Examine operational documentations, observe people when they are at work, or apply process model discovery algorithms to event logs;
- Interview-based: Interview domain experts about the process. This method offers a rich and detailed picture of the process. However, the interview-based method is labour-intensive because multiple iterations are often needed;
- Workshop-based: Organise a process modelling workshop that includes process analyst and domain experts. This brainstorming session allows varying ideas to be formed and challenged at the same time. A good facilitator is required in workshop-based techniques.

Process modelling techniques often comprises of the following steps:

- Look for the boundary of a process by identifying the start and end events;
- Identify main activities;
- Identify main process participants and their handover;
- Determine the control-flow of a process;

- Capture information artefacts and exception handlers. Annotations are also added in this step.

2.1.5 Process Analysis - Qualitative Methods

Process analysis aims at identifying which part of a business process has a problem or needs improvements. Qualitative process analysis methods include a few basic principles and techniques, such as value-added analysis, root cause analysis, and impact assessment.

In value-added analysis, unnecessary activities that do not offer values to customers or business are identified and eliminated. In root cause analysis, the process analyst identifies the root cause(s) of a problem or an undesirable event happened during the execution of a business process. The outputs of this method, for example, are (i) cause-effect diagrams, which can be found by identifying the causal or contributing factors, (ii) why-why diagrams, which can be found by recursively asking questions until the root cause is identified. In the impact assessment method, issues are documented and prioritised so that the most severe ones would be solved first. *Pareto* analysis (Karuppusami and Gandhinathan, 2006) and PICK charts (Dumas et al., 2013) are then used to identify a small set of factors that cause the most problems.

2.1.6 Process Analysis - Quantitative Methods

In quantitative methods, process performances are measured by a set of criteria, some of which are shown below:

- Cycle time (throughput time, flow time, lead time) of a case is the time measured from the beginning to the end of that case.
- Processing time (service time) of a case is the actual time that resources spend on handling that case.
- Waiting time (idle time) of an activity/case is the time when no resources are available for that activity/case.
- Synchronisation time of an activity is the time that a resource waits for the completion of another activity or an external trigger.
- Costs, e.g. production cost, delivery cost, labour cost.
- Customer satisfaction level.

Balanced scorecard (Kaplan and Norton, 2005) is a quantitative method that measures performances of an entire organisation instead of a single business process. The main

performance indicators used in this method are categorised into four dimensions (Dumas et al., 2013), e.g. finance measure, internal business measure, innovation and learning measure, customer measure.

References models are sets of standardised performance measures for common types business process. For example, in the *Supply Chain Operations Reference Model (SCOR)*, the measure *Purchase Order Cycle Time* is defined as the *average amount of time between the moment an intention to purchase is declared and the moment the corresponding purchase order is received by the relevant vendor* (Dumas et al., 2013).

Flow Analysis

In flow analysis, the overall performance of a process is estimated based on the performance of its individual activities and the chance of them being executed, which is in turn based on the control-flow aspect of the process model. For example, given the occurring probabilities of activities in a process model, the average cycle time of a process can be calculated from the average cycle time of each activity.

Other tasks in flow analysis include calculating cycle time efficiency, arrival rate, and work-in-progress. The cycle time efficiency of a process is equal to the fraction of processing time over cycle time. The arrival rate of a process is the average number of new instances that are created per time unit. The work-in-progress is the average number of instances that are active at a given point in time.

Queue Analysis

Flow analysis does mention about resource contention which often occurs in systems where resources are limited. Queuing theory (Gross and Harris, 1998) can be used to estimate the expected waiting time of a case.

Simulation

Many organisations have used simulation to analyse their business processes (van der Aalst, 2011a). The idea of simulation is to repeatedly generate hypothetical instances of a business process using computer models. The generated instances, which are in large number, are then used to build statistics on average flow time, waiting time, costs, etc.

2.1.7 Process Redesign

The process analysis phase explores weak points of a business process, which are then resolved in the process redesign phase.

There are many techniques that can be used in the design process, such as fish-bone diagramming, *Pareto* analysis, Delphi method (Linstone and Turoff, 1975), flowcharting, etc.

A business process can be redesigned by three approaches (Dumas et al., 2013): (i) the existing processes are redesigned from scratch, (ii) the existing processes are extended or modified, and (iii) processes are redesigned based on reference models, which are standard solutions made by consulting companies. Of the three, the process extension and modification approach is the most popular.

Process redesign can start with an examination of the process from different perspectives such as the customer perspective, the organisational structure perspective, and the technology perspective. For example, from the customer perspective, receiving documents through mail service may cause long waiting time, attaching documents to electronic emails should be used instead.

In the *Product-based Design* approach (Reijers, 2003), product specifications are decomposed into a set of interrelated parts. A product data model is then built based on the identification of the logical dependencies among these parts. Finally, the process model is then derived from the product data model and design goals.

In redesigning of a business process, the four process performance dimensions, i.e. time, cost, quality, and flexibility, have to be continuously evaluated (Dumas et al., 2013). Although the aim is to maximise all of the four dimensions, trade-offs are often made. For example, the automation of business activities would reduce the flow time of a process, however, the equipment and training cost would probably increase.

As mentioned in Section 2.1.4, the product of the process redesign phase are to-be process models, which become inputs to the next phase, i.e. business process implementation/configurations.

2.1.8 Business Process Implementation/Configurations

Business processes are implemented based on a set of policies and procedures that the employees of the organisation need to comply with (Weske, 2007). In case an automation for business processes is needed, a *BPMS* is built based on the to-be process models developed in the previous phase and the organisation environment. Integrations with legacy systems are also configured in this phase.

The transformation of abstract process models into executable programs is described in five steps (Dumas et al., 2013), which are as follows.

- Identify the automation boundaries. Because not all parts of a business process can be automated, it is necessary to define the boundary of each task where the automation is still applicable. In addition, manual and user tasks are also needed

to be specified. In manual tasks, no software is involved, and in user tasks, there is an coordination between the system and users.

- Review manual tasks. The manual tasks are examined if they can be automated partly in *BPMS*.
- Complete the process model. In this step, any gap between the business-oriented models and the executable process models is checked.
- Bring process model to an adequate granularity level. The decision to join or split tasks is based on the workload of resources or the complexities of the task.
- Specify execution properties. In this step, how each element in the process model is implemented is specified. For instance, the executable script of an activity in a process model needs to be specified in this step.

After the system is configured, the implementation of the business process is evaluated. The system is then deployed to its target environment. Interactions between the process participants and the information systems are changed accordingly to the re-designed processes. Therefore, a change management and employee training program are needed.

The main benefit of automating a business process is that the workload of process participants is significantly reduced. This is because information is transferred instantly, automatic work dispatching software reduces delays in job handover, and the amount of information exchanges is reduced due to the data sharing architecture. In addition, a *BPMS* increases the transparency of the system and allows business rules to be automatically enforced.

2.1.9 Process Enactment

The process enactment starts once the information system is deployed and properly configured. In this phase, process instances are executed to fulfil the business goals of a company. Process execution data, which includes events, resources, activities, etc., are archived in event logs for further analysis.

2.2 Process Mining

The process mining phase comes after the process enactment phase. It focuses on the analysis of event logs, and from its results prompt adjustments to the process can be made. In addition, the output of process mining is useful for the redesigning phase in the next iteration.

Many related fields falls under the umbrella of process mining such as *Business Process Intelligence (BPI)*, *Business Activity Monitoring (BAM)*, *Complex Event Processing (CEP)*, *Process Intelligence (PI)*, *Process Analysis (PA)*, and *Business Intelligence (BI)*. *PI* and *BPI* essentially bear the same meaning with Process Mining. However, these terms are loosely defined and used arbitrarily by different vendors. The field *BAM* and *CEP* refer to the monitoring of a stream of events; any policy violations, or performance bottleneck are reported to the process participant or the process owner (a manager responsible for a business process). The term *PA* describes analysis activities whose goal is to improve business processes. While process mining revolves around the notion of process, *BI* refers to the general application of data mining to business data.

Process mining techniques have been applied to many domains such as in health-care (Mans et al., 2009), manufacture (Rozinat et al., 2009a), public services (Bozkaya et al., 2009; Rozinat et al., 2009b), financial services (Jans et al., 2011; Sonnenberg and vom Brocke, 2014), and telecom industry (Goedertier et al., 2011).

There are three main categories of process mining tools: 1) commercial software (e.g. *ARIS Process Performance Manager*¹, *DISCO*², *Interstage Process Discovery*³, *Celonis Process Mining*⁴), 2) academic tools (e.g. *Genet/Petrify* (Carmona et al., 2009), *Rbminer/Dbminer* (Solé and Carmona, 2010)), and 3) open source tools (e.g. *PROM* (van Dongen et al., 2005)).

One of the first process mining tools was *Business Process Cockpit (BPC)* (Sayal et al., 2002). This program supports real-time monitoring, analysis, management, and optimisation of business processes. The *PISA* (Muehlen and Rosemann, 2000) software calculates performance metrics from workflow logs. Process miner (Schimm, 2003) implements a π -calculus algorithm over a block-structured model and claims to be able to mine the fittest model. Stochastic graph algorithm is used in *InWolve* (Herbst and Karagiannis, 2004). *PROM* (van Dongen et al., 2005) is one of the prominent software frameworks for process mining. This plug-in-based framework allows for the add-on of user-developed process mining algorithms. Several examples of *PROM* plug-ins are: (i) *Emit* (van der Aalst and van Dongen, 2002), a process discovery method that realises the α -algorithm (Van der Aalst et al., 2004), (ii) *LittleThumb* (Weijters and van der Aalst, 2003), another process discovery method that uses heuristic mining algorithm, and (iii) *Mison* (Van Der Aalst et al., 2005), a social network miner that explores the hand-over of work among process participants.

There have been several other process mining framework, which are as follows.

¹<http://www.softwareag.com>

²<http://fluxicon.com/disco/>

³<http://www.fujitsu.com/>

⁴<http://www.celonis.de/>

- In (Grigori et al., 2004), the authors present a complete architecture for the organizing, storing and loading process data. It also features functionalities such as prediction, monitoring, control and optimisation of process executions. The proposed architecture lays the ground for the development of process mining applications. Many examples of applying data mining methods to process data was given, however, no particular algorithm, nor method was proposed.
- In (Bozkaya et al., 2009) the author proposed a six-phase process diagnostics methodology which includes *log preparation*, *log inspection*, *control flow analysis*, *performance analysis*, *transfer results*, and *role analysis*. Specifically, *role analysis* can be performed at any time after *log preparation*. The focus of this study was also to provide a general procedure for process log analysis, without giving specific details of any algorithm or method.
- In (Rebuge and Ferreira, 2012), the authors extend the work of (Bozkaya et al., 2009) by adding a *Sequence Clustering Analysis* phase after the *log inspection* phase and the method is used for the health care domain. The clustering method uses first-order *Markov* chains to represent each cluster. The *Expectation-Maximisation* (Han and Kamber, 2006) is used to discover clusters. The cluster with highest number of instance contains the regular behaviour of the process. The minimum spanning tree algorithm is then used to suggest related clusters containing the infrequent behaviours of the process. The proposed methodology focuses on clustering and outlier detection of process instances based on the control-flow of the process instances without taking into account other data attributes. Furthermore, it is not clear if the *Markov*-based method is scalable in processes having many activities/events.
- In (De Weerd et al., 2013), the authors argue that real-life process analysis should be done from multiple perspectives, i.e. control-flow, team flow, and document flow. It suggests a process mining framework consisting of five phases: *preparation*, *exploration*, *perspectivisation*, *analysis* and *results*. This framework treats activities, document types, and actors separately; this might limit some patterns that can only be discovered if the data is treated as a whole. Domain knowledge is usually required to aggregate and interpret patterns gleaned from multiple perspectives.

The process mining framework suggested in (van der Aalst, 2011a) is one of the most accepted methodologies for process analysis. This framework was briefly introduced in Chapter 1 and more details are described in the remainder of this section. Event log and related concepts are presented in Section 2.2.1, 2.2.2, 2.2.3 and 2.2.4. Process model discovery is discussed in Section 2.2.5.

2.2.1 Event Logs

Although event logs are available in most information systems, they often reside in different databases, with different data structures. For example, it is not uncommon that each department in a company uses a different database system, e.g. an accounting department uses a proprietary accounting software, while the sales department uses a customer relationship management program. In a global company, the data are even from different sub-companies.

The data are stored in a variety of format such as databases, files, document management systems, etc. In many cases, the data are unstructured such as images, emails, or *PDF* documents, which is difficult to extract relevant information for process mining algorithms.

Apart from the two mentioned issues, there can be semantic issues or format issues. For example, in one data source, the field name is *movie*, however, and in another data source, the corresponding field is named as *film*.

Date/time is a common format mismatch issue. For instance, date/time data can be represented as “Day/Month/Year” or “Month/Day/Year”.

Another common issue is the scale of information systems. In many cases, there are thousands of tables that store event data. To reduce the complexity of the data extraction process, process mining goals have to be identified in the first place. Generally, the data extraction process is usually involved in three tasks: extraction, transform, and load. The extraction task aggregates data from different data sources. The transform task solves the structure mismatches between the data sources and the event log, e.g. removing unnecessary columns in the data sources, or aggregating values from multiple rows. The load task is to populate the transformed data into event logs.

2.2.2 Event Log Structure

During the execution of a business process in a *BPMS*, whenever a task/activity starts or completes, an event is generated and logged into the system. Besides event, other data are also recorded. However, process mining typically focuses on activities name, resources—persons who carry out the task/activity, timestamps when the events occur, the costs of carrying out the event.

Since many processes can be running at the same time in an organisation, the events are mixed together in different data sources. It is required that an event log only contains events belonged to a single process.

A general structure of an event log contains multiple process instances, events, and their attributes (van der Aalst, 2011a) such as activity and timestamp. A process instance starts at the time a business process starts and ends at the time that process ends. Process instance is also known as case, which contains a set of events. The

Table 2.1: An event log for a conference reviewing process.

Case	Event	Timestamp	Activity	Resource	Lifecycle	Cost
173	39518	2019-02-11	Invite Reviewers	Mary	start	15
173	39519	2019-02-13	Invite Reviewers	Mary	complete	15
173	39520	2019-02-14	Get Review	Pete	start	2
173	39521	2019-02-18	Get Review	Pete	complete	2
174	41922	2019-02-21	Invite Reviewers	Bob	start	19
174	41923	2019-02-22	Invite Reviewers	Bob	complete	19
174	41924	2019-02-25	Invite More Reviewers	Bob	start	11
174	41925	2019-02-28	Invite More Reviewers	Bob	end	11

events are ordered chronologically based on their *timestamps* (the time when the events occur). The activity attribute is a mandatory attribute of each event, which represents the name of the task performed in a process. An ordering of activities of a case based on their corresponding timestamps is called a trace.

Finally, each event can have additional attributes such as cost, resource, transaction, etc. The timestamp attribute is the time when an event occurs. The cost attribute is the cost associating with the task performed. The resource attribute is the human or software that performs the task. The transaction attribute describes the status of the task, e.g. start, assign, abort, etc. Note that events that share the same activity have the same set of attributes.

Table 3.4 gives an example of an event log for a conference reviewing process. There are two cases with ID 173 and 174. Each case is composed of four events with different event IDs and timestamps. For case ID 173, there are two activities “Invite Reviewer” and “Get Review”. The “Invite Reviewers” activity was done by “Mary” with the cost of 15. Note that each activity has two transaction values, i.e. *start* and *complete*, which represent the starting event and the completing event of the activity.

2.2.3 Notation

Let \mathbb{E} be the set of all events. Each event may have one or more attributes. Every event has an attribute ID. For each event $e \in \mathbb{E}$, $A_n(e)$ is the value of the attribute n of event e . For instance, $A_{activity}(e)$ is the activity name of event e .

Let \mathbb{C} be the set of all cases. For each case $c \in \mathbb{C}$, $A_n(c)$ is the value of attribute n for case c . Trace is a mandatory attribute of case, with $A_{trace}(c) \in \mathbb{E}^*$.

In a trace, each event is unique, i.e. in a case c , for any $i \neq j$ and $i, j < |A_{trace}(c)|$: $(A_{trace}(c))_i \neq (A_{trace}(c))_j$. Note that $|A_{trace}(c)|$ is the number of events in the trace of case c and $(A_{trace}(c))_i$ is the i^{th} event in trace $A_{trace}(c)$. The set of events in case c is denoted by \mathbb{E}_c . The order of events in a trace depends on the timestamp of the events, i.e. for any $1 \leq i < j \leq |A_{trace}(c)|$: $A_{time}(A_{trace}(c))_i < A_{time}(A_{trace}(c))_j$

In an event log, each event is unique, i.e. for any $e_1, e_2 \in \mathbb{C}$: $\mathbb{E}_{e_1} \cap \mathbb{E}_{e_2} = \emptyset$.

2.2.4 XES

As mentioned in Chapter 1, *XES* is the successor of *MXML*—an *XML*-based language that is used to represent event log. *XES* is more flexible than *MXML* in that it does not restrict a fixed set of attributes for each event.

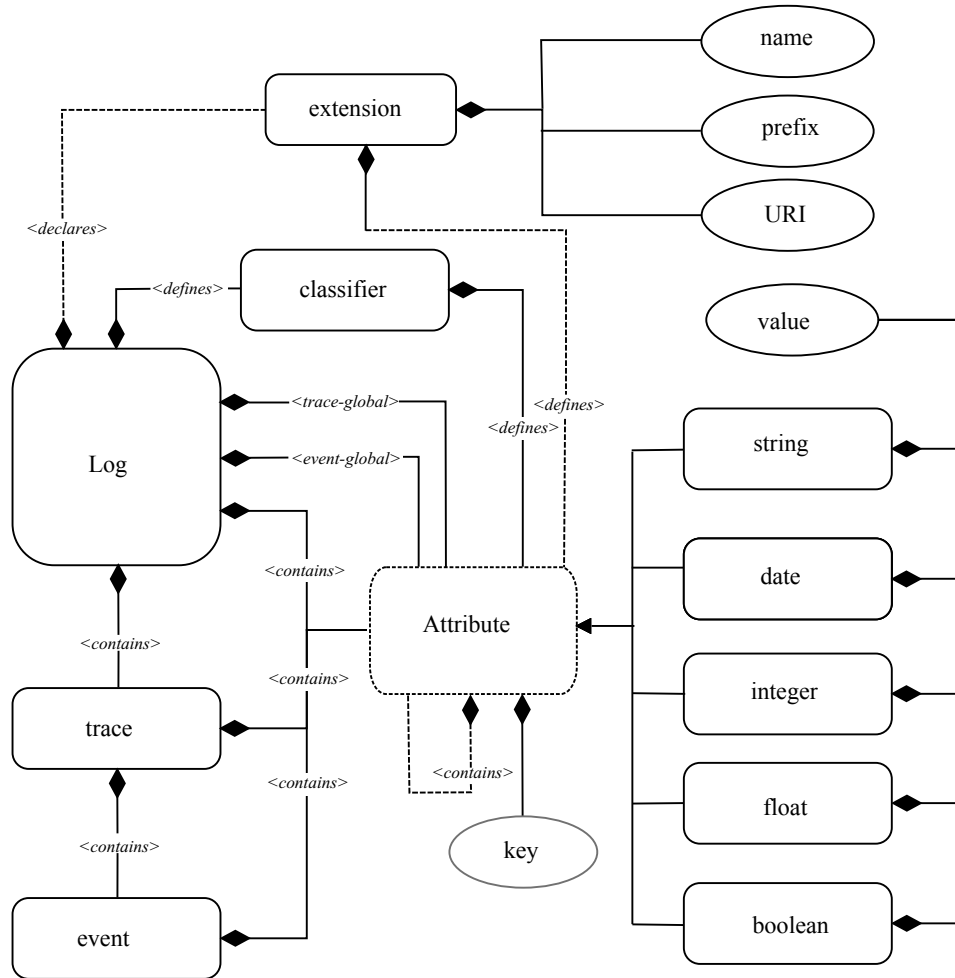


Figure 2.2: *XES* meta model (Verbeek et al., 2011).

The meta model of *XES* is displayed in Fig. 2.2 and an example of an event log in *XES* is shown in Listing 2.1. It is seen in the model that an event *log* contains one or many *traces*, which in turn contains one or more *events*. Each of these elements could contain one or more *attributes*, which can be nested. Each attribute has two mandatory components, i.e. *key* and *value*. Five main data types of attribute values are *String*, *Float*, *Int*, *Date*, *Boolean*. These correspond to *XML* types *xs:string*, *xs:long*, *xs:double*, *xs:dateTime*, and *xs:boolean*. A log may have one or more classifiers, each of which declares attributes that identify an event. For instance, an event can be identified by its activity and life cycle transition (*start*, *end*, etc.). The extension mechanism

in *XES* provides definitions for the *key* of attributes. An *extension* contains *names*, *prefix* and *URI*. *Prefix* is an abbreviation of the extension name. An *URI* contains the location where the extension was declared. There are five extensions defined in *XES*, which are shown as follows:

- The concept extension contains the name attribute which is used to define the name of traces and events;
- The life cycle extension contains the transition attributes such as *schedule*, *start*, *complete*, etc., that are used to describe the status of an event;
- The organisational extension contains attributes that describe a resource such as resource, role, and group;
- The time extension contains the time attribute for events;
- The semantic extension contains the *modelReference* attribute. Ontologies, which describe the concepts used in the event log, are declared in the *modelReference* attribute.

Listing 2.1: A fragment of a *XES* document

```
<?xml version="1.0" encoding="UTF-8" ?>
<extension name="Concept" prefix="concept"
  uri="http://concept.xesext"/>
<extension name="Organizational" prefix="org"
  uri="http://org.xesext"/>
<extension name="Time" prefix="time" uri="http://time.xesext"/>
<global scope="trace">
  <string key="concept:name" value="name"/>
</global>
<global scope="event">
  <date key="concept:name" value="name"/>
</global>
<classifier name="Activity" keys=concept:name"/>
<classifier name="Resource" keys=org:resource"/>
<trace>
  <string key="concept:name" value="3571"/>
  <string key="description" value="Simulated process instance"/>
  <event>
    <string key="org:resource" value="Anne"/>
    <date key="time:timestamp" value="2054-05-02T01:00:00.000+02:00"/>
    <string key="concept:name" value="invite reviewers"/>
    <string key="lifecycle:transition" value="start"/>
  </event>
</event>
```

```
<string key="org:resource" value="Anne"/>
<date key="time:timestamp" value="2054-05-04T01:00:00.000+02:00"/>
<string key="concept:name" value="invite reviewers"/>
<string key="lifecycle:transition" value="complete"/>
</event>
<event>
  <string key="result" value="reject"/>
  <string key="org:resource" value="Mary"/>
  <date key="time:timestamp" value="2054-05-05T01:00:00.000+02:00"/>
  <string key="lifecycle:transition" value="complete"/>
  <string key="concept:name" value="get review 1"/>
</event>
<event>
  <string key="result" value="reject"/>
  <string key="org:resource" value="Sara"/>
  <date key="time:timestamp" value="2054-05-06T01:00:00.000+02:00"/>
  <string key="lifecycle:transition" value="complete"/>
  <string key="concept:name" value="get review 2"/>
</event>
</trace>
```

The quality of the event log is an important factor contributing to the success of a process mining project. There are many challenges in obtaining a good quality event log. There can be uncorrelated events in the event log because aggregating event data from different data sources is an error-prone process. However, by using a probabilistic method, the study in (Ferreira and Gillblad, 2009) is able to identify events that belong to different business processes. Other challenges and their corresponding solutions in obtaining quality event data are described in (van der Aalst, 2011a). Among those discussed, one common challenge is to deal with information systems that have thousands of tables. It is hard to identify which tables to extract. One typical solution is to use domain knowledge to locate the required data and/or identifying specific goals in early phases of the process mining project.

2.2.5 Process Model Discovery

Process models are used in most phases in a *BPM* life cycle, therefore learning how process models are built is an important topic. The five-step process modelling techniques (see Section 2.1.4) is one of the most popular methods of making process models. However, since the advent of event logs, there is a shift of focus from modelling to discovering processes.

The process models discovered from process logs offer valuable information on how a business process works in reality. Most process model discovery algorithms focus on the control-flow perspective of a business process. From this point forward, the

phrase process discovery implies the control-flow perspective of the process discovery.

Process models can be discovered from event logs using techniques inherited from the data mining and machine learning fields. A recent survey on different process mining algorithms was published in (Tiwari et al., 2008).

The process discovery problem was first considered a grammar inference problem (Gold, 1967). However, regular expression is the least expressive language and less helpful for business process management.

Algorithmic approaches extract ordering relations among activities to form Petri net models. The earliest and most well-known algorithm is α -algorithm (Van der Aalst et al., 2004) which simply identifies the relations between each pair of activities in the event log. Based on the relations found, the algorithm identifies the resulting model in Petri net representation. $\alpha+$ algorithm (de Medeiros et al., 2003) is an improvement over α -algorithm in handling short loops. In multi-phase miner (van Dongen and van der Aalst, 2004), instance graphs, a simpler form of process model, are created for each instance and then aggregated into a process model. This method can cope with noisy event logs.

Directed acyclic graphs are used to represent a process model in (Agrawal et al., 1998). The presented method identifies a dependency graph from the log and only detects sequential models and non-duplicate tasks. Frequent closed partial orders are searched in sequential data to discover parts of the process model (Pei et al., 2006). Depth-first search is used to find all frequent partial orders in the form of transitive reduction representation. Frequent partial orders are also searched in (Pei et al., 2006) but it is limited to the finding of series-parallel orders.

Probabilistic models (neural network based) and algorithmic approaches, e.g. *Ktail* algorithm (Cook and Wolf, 1998a), are used to detect sequential process model. The authors proposed a hybrid approach which uses n-order Markov model and yields better results. This technique is later improved in (Cook and Wolf, 1998b) to deal with concurrency. Finite State Machine (Hopcroft and Ullman, 1979) is used in (Datta, 1998), but solely based on a probabilistic method. Probabilistic methods are also used to decompose events into independent subsequences (Mannila and Rusakov, 2001).

The studies in (Herbst and Karagiannis, 2004; Herbst, 2000; Herbst and Karagiannis, 1998) are among the first to examine both the sequential/concurrency and the duplicate tasks issue. It uses a stochastic graph as an intermediate representation to construct a process model based on the *ADONIS* language. *Hidden Markov Model* is used to predict the stochastic graph which is then merged or split (similar to (Cook and Wolf, 1998a)).

Another class of approaches use negative event samples to discover models (Goedertier et al., 2009; Lamma et al., 2007). Prior knowledge is used as a valuable guide for process discovery algorithm (Ferreira and Ferreira, 2006). The *AGNE* method (Bloc-

keel, 1998) allows user-specified first-order representation of prior knowledge, which is then used to generate negative events. This added information helps solve the incompleteness issue in process mining. Positive and negative samples are fed into a classification algorithm, called *TILDE*, in order to extract a set of preconditions among the activities, which can be later converted to Petri net models. *DecMiner* (Lamma et al., 2007) is a similar method that uses real negative events and induces a *SCIFF* model (Alberti et al., 2008), which is later converted to *DecSerFlow* declarative graphical language.

Parallel activities in process models are hard to detect in event logs. Finite state machine methods perform well on outliers but have problems in recognizing parallel actions. On the contrary, the α -algorithm cannot deal with outliers but is better at detecting concurrency. Two *Markov-chain*-based algorithms, *RNet* (neural-network-based) and *Ktail* (Cook and Wolf, 1998a), do not support concurrency. This problem was solved in the later work (Cook and Wolf, 1998b).

Besides the main approaches mentioned above, many other approaches were also proposed. In a transition system approach (van der Aalst et al., 2010), a low-level model similar to finite state machines is created and then converted to higher level model (*Petri net*, *BPMN*, *YAWL*, etc.) using a theory of regions (Cortadella et al., 1998).

To detect usage scenarios in a process model, event logs are clustered using discriminating rules (Greco et al., 2006) (each discriminating rule is considered a feature in the clustering model). Dependency graph creating techniques (Agrawal et al., 1998) are also used in this method.

The three *Apriori*-based algorithms (Agrawal and Srikant, 1994), *TP-Graph*, *TP-Itemset* and *TP-sequence*, are developed to detect frequent temporal patterns in event logs (Hwang et al., 2004).

A robust decision-tree-based classifier is used to identify the relation types between each pair of activities in event logs (Mărușter et al., 2006).

The *Schimm* method (Schimm, 2003) represents process models in a block-structured form and uses term rewriting rules for model transformation.

A fuzzy mining method was suggested in (Günther and van Der Aalst, 2007), where different techniques such as edge filtering and node aggregation can be used to simplify the learned process model.

Evolutionary approaches are robust but computationally expensive (de Medeiros et al., 2007). The work of (Hwang et al., 2004) presented a method that is similar to genetic algorithms. In this method, different combinations of activities and constructs in a tree representation are evaluated and a branch that has a bad *mismmerge* score will not be further explored.

2.2.6 Main Approaches to Process Log Analysis

This section discusses two main approaches to event log analysis: one is based on process models and the other is not.

To have an estimate of the proportion of papers that focuses on process model discovery, the *Google Scholar*⁵ search engine was used to search for papers containing the phrase “process mining”. The first 100 papers having the most citations was analysed. The results were divided into 10 categories and their frequencies are displayed in Table 2.2. It can be seen that 69 out of 100 papers are based on process models, whereas only one method uses direct data mining technique. This clearly shows that lion’s share of process mining research has been devoted to analysis techniques that are based on process models.

Table 2.2: Categorisation of the process mining literature.

ID	Category	Frequency
1	Survey and review	11
2	Book	2
3	Process model based analysis	69
4	Semantic-based analysis	3
5	Simulation/Visualisation	2
6	Pre-processing techniques	2
7	Declarative models	2
8	Data mining techniques	1
9	Miscellaneous	2
10	Tool	6

Data mining techniques have been used widely in the system log analysis. An example of a typical system log file is shown in Listing 2.2. It can be seen that the activities in the log are automatically generated by computer programs. This is in contrast to process logs where activities are performed by either human or machine; both are guided by process models. As a result, two different research communities exist: one is associated with mining process logs and the other is related to mining system logs. Some notable works on mining system logs are presented in (Makanju et al., 2012; Vaarandi, 2004; Xu et al., 2009; Sipos et al., 2014).

Listing 2.2: An example of a system log file (reproduced from (Makanju et al., 2012))

```
2005-06-03-15.42.50.823719 R02-M1-N0-C:J12-U11
RAS KERNEL INFO instruction cache parity error corrected
2005-06-03-15.42.50.982731 R02-M1-N0-C:J12-U11
RAS KERNEL INFO instruction cache parity error corrected
2005-06-06-22.41.37.357738 R20-M0-NA-C:J15-U11
RAS KERNEL INFO generating core.3740
```

⁵<http://scholar.google.com.au>


```

2005-06-06-22.41.37.392258 R20-M0-NA-C:J17-U11
RAS KERNEL INFO generating core.3612
2005-06-11-19.20.25.104537 R30-M0-N9-C:J16-U01
RAS KERNEL FATAL data TLB error interrupt
2005-06-11-19.20.25.393590 R30-M0-N9-C:J16-U01
RAS KERNEL FATAL data TLB error interrupt
2005-07-01-17.52.23.557949 R22-M0-NA-C:J05-U01
RAS KERNEL INFO 458720 double-hammer alignment exceptions
2005-07-01-17.52.23.584839 R22-M0-NA-C:J03-U01
RAS KERNEL INFO 458720 double-hammer alignment exceptions

```

The number of studies that focus on mining event data without process model is rather limited. The authors in (Maggi et al., 2014) proposed a method that uses a decision tree learning algorithm to predict process instances that are high risk. The method recommends activities and/or attributes to be executed and/or selected, respectively, in order to maximise the probability of fulfilling a business constraint. Note that these constraints can be written in linear temporal logic rules.

The work in (Nakatumba et al., 2012) studied the relationship between workload and service time by using linear regression analysis.

Suriadi et al. (2013) introduced a method that is able to identify root causes of risk incidents, such as process instances that are overtime. Event logs are enriched and converted into a suitable format for a decision tree learning algorithm. The features used by the algorithm are decided by the users. Note that process models are not consulted in any way.

The study in (Vasilyev et al., 2013) used *Inductive Logic Programming* (Lavrac and Dzeroski, 1993) to identify causes of process delays. The data are converted into first order logic representation. Then, hypotheses are learned using a decision tree learning algorithm. It can be seen that no process models were used in the process.

Aside from the two main approaches, there are methods that use process model in a limited way. In (Pika et al., 2013), a set of *Process Risk Indicators (RPI)* (e.g. waiting time) for predicting case delays were proposed. Process instances whose *RPIs* are higher a predefined-threshold are predicted as “delay”. Based on the analysis of historical event data, the method identifies a threshold for each *RPI*. Of all *RPIs*, a few of them, such as waiting time and sub-process duration, are estimated based on process models. In (Rozinat and van der Aalst, 2006), the authors suggest that attributes of the activities executed right before a decision point in a process model can be used as feature variables, and alternative routes after the decision points can be used as class labels in a supervised learning algorithm.

Generalised stochastic *Petri nets* (an enhanced form of *Petri net*) were used to predict remaining process execution times in (Rogge-Solti and Weske, 2013).

Rozinat and van der Aalst (2006) introduced a method that is able to predict next

activities from a current activity, which is based on a known process model.

A risk-aware process mining approach was suggested in (Conforti et al., 2013). Whenever a decision point is reached during a process execution, a decision tree learning algorithm is used to predict different types of risks, e.g. overtime and cost overrun. Note that a process model might be consulted to identify decision points.

Summary

A plethora of frameworks, tools and algorithms have been developed for process mining. Most studies on mining process logs focus on process discovery or utilise process models for further analysis such as conformance checking, simulation and model enhancement. The reason for this unbalance is probably due to the fact that process models are intuitive for users.

There are few methods that do not use process models but they often explore event data from a single perspective, i.e. selecting only one or several attributes, such as time or cost, for the data analysis. In general, there is a lack of process mining methods that can be applied directly to event data, especially *XML*-based process log. In the following sections, data mining methods and concepts are introduced.

2.3 Related Data Mining Concepts

This section introduces basic data mining concepts and notations, which are used in subsequent sections. Essentially, data mining, or knowledge discovery from data, “is a process of discovering interesting patterns and knowledge from large amounts of data” (Han and Kamber, 2006). Data mining tasks can be generally divided into two main categories: descriptive and predictive mining. The descriptive mining tasks identify characteristics/properties of a data set, such as frequent pattern mining, association rule mining and clustering analysis. The predictive mining tasks construct induction models from the training data to predict future/testing data, such as classification, regression analysis and outlier analysis.

Section 2.3.1 describes data types that are related to the process mining of data. Section 2.3.2 introduces the frequent pattern mining field, which concerns about finding substructures that occurs many times in a database. Sections 2.3.3, 2.3.4, 2.3.5, and 2.3.6 subsequently describe specific types of frequent patterns. Closed and maximal patterns are two condensed forms of frequent patterns, which are explained in Section 2.3.7. Section 2.3.8 presents general approaches for identifying frequent patterns in a database. Association rule mining, an unsupervised and descriptive data mining task, which is able to discover interesting relationships in event logs, is presented in Section 2.3.9. Classification, a supervised and predictive data mining task, which

is able to predict the outcomes of future data, is presented in Section 2.3.10. Section 2.3.11 discusses a number of measures that are used to evaluate a classification algorithm. Section 2.3.12 introduces several methods that are used to obtain reliable results from classification. Associative classification is a promising approach of unsupervised learning methods, which is explained in Section 2.3.13. Lastly, Section 2.3.14 is dedicated to interestingness measures, which are used to assess whether a pattern is interesting or not.

2.3.1 Types of Data

In this thesis, different data types are used, i.e. semi-structured, flat, itemset and tree-structured data. Therefore, the following subsections provide an overview to common data types.

Structured and Relational Data

Structured data are data that strictly follow a predefined schema (Sint et al., 2009) such as spreadsheet, relational data, etc. A relational database comprises of a set of tables, each of which is referred to as a relation and assigned a unique name. The columns of a table represent attributes/fields and the rows of a table represent instances. A tuple (instance) is a set of values, each of which corresponds to a column of the table. Relational data is also called flat data, or flat representation. Relational data is a common data type used in many *BPMS* and software systems nowadays. Moreover, the analysis/mining of relational data is well-studied in the data mining field. van der Aalst (2011a) introduced a method to transform relational data a format that is suitable for process mining algorithm.

Semi-structured Data

There have been no precise definitions for semi-structured data and they are often considered as data with no rigid structure, or data that are neither raw nor strictly typed (Abiteboul, 1997). For example, a *BibTex* (Beebe, 1993) file contains data with structures resembling relational data, however, there are other properties that a well-structured database does not have such as missing mandatory fields, or cross references. For this reason, *BibTex* files are considered as semi-structured data.

The Web is the area where there are many examples of semi-structured data such as *XML*, *HTML* (Raggett et al., 1999), *SGML* (Goldfarb and Rubinsky, 1990), etc. This can be explained by the fact that the Internet has numerous types of data, and a flexible language, such as semi-structured data, is required for data integration and exchange. Mining semi-structured data from the Web is an active field of research (Madria

et al., 1999; Miyahara et al., 2001; Hovy et al., 2013). Semi-structured data has been spreading to many domains such as biology (Deepak et al., 2014), scientific management (Cooper et al., 2001), chemistry (Dehaspe et al., 1998), etc. General algorithms for mining semi-structured data were studied in (Arimura et al., 2001; Asai et al., 2002).

Many semi-structured data such as *XML*, proteins, and *DNA* can be expressed as tree structures or graphs (Shahbazi and Miller, 2014; Yan and Han, 2002; Huan et al., 2003; Chi et al., 2004b). Tree-structured and graph-structured data are fundamental data structures. In a tree-structured database, each instance is a tree which comprises of a root, a set of nodes and arcs connecting them. Trees are acyclic, meaning that no path (a set of connected arcs) starting from one nodes and returning to that same node is allowed. In a graph-structured database, each instance is a graph which contains a set of nodes and arcs connecting them. Graphs can be cyclic, meaning that paths running through the same node twice or more are allowed.

Unstructured Data

Unstructured data is the most common type of data (Blumberg and Atre, 2003). Examples of unstructured data are text documents, audio, video, email, notes, etc. According to (Sint et al., 2009; Hadzic et al., 2011b), unstructured data have no defined schema or the form of structure is not useful for a desired task. There have been efforts in mining process models from textual data, but the result is still limited in mining process description documents (Friedrich et al., 2011).

Other Types of Data

Itemset data include a set of transactions, each of which contains a set of items. Itemset data mining was inspired from market basket data analysis. Market basket data refer to a transactional database which comprises of items purchased by customers (Agrawal and Srikant, 1994). This database includes a number of transactions, each of which contains a set of different items. From market basket data, associations among items purchased can be discovered. This information can be used to decide which items should be put on sale, or placed near each other on shelves. The frequent itemset mining on market basket data is introduced in Section 2.3.3.

As categorised by (Han and Kamber, 2006), three main types of sequence data are time series data (e.g. stock exchange data), symbolic sequences (e.g. customer shopping sequences, web click streams), and biological sequences (e.g. *DNA* and protein sequences). Sequential data mining is described in Section 2.3.4.

2.3.2 Frequent Pattern Mining

Frequent patterns are defined as “itemsets, subsequences, or substructures that appear in a data set with frequency no less than a user-specified threshold” (Han et al., 2007b). For instance, if beers and diapers are often found together in a transactional database of a supermarket, they are frequent itemsets. A subsequence, such as navigating to a homepage, then a product page, and then a specific product (usually found together in a website access history database), is an example of (frequent) sequential pattern. A substructure is a general term that describes many types of structural units, e.g. subgraphs, subtrees, sublattices. Substructures that often appear together in a database are called (frequent) structural patterns.

Frequent pattern mining plays an important part in the data mining field (Aggarwal and Han, 2014). It helps identify associations, correlations, and other relationships in data, as well as in data indexing, classification, clustering, etc. (Han et al., 2007b). In the following subsections, different types of structural pattern are discussed.

Formal Definitions

One important concept in frequent pattern mining is *support*, which is the ratio of transactions in which a substructure appears to the total number of transactions. Note that the word ‘appear’ can be defined in a number of ways; for example, a subtree can appear in a tree either in an ‘induced’ or ‘embedded’ manner (described later in this chapter).

Definition 2.1 (Support). *Let X be a substructure in a database that has n transactions, and m is the number of transactions where X appears, the support of X , $\pi(X)$, is defined as follows*

$$\pi(X) = \frac{m}{n}.$$

Definition 2.2 (Absolute Support). *Let X be a substructure in a database that has n transactions, and m is the number of transactions that X appears, the absolute support of X , $\pi^A(X)$, is defined as follows*

$$\pi^A(X) = m.$$

In frequent pattern mining, a substructure (X) is considered *frequent* if its support is higher than a user-specified threshold (minimum support threshold) π .

Definition 2.3 (Frequent pattern). *The set of frequent patterns (\mathcal{F}) with a minimum support of π in a database is defined as:*

$$\mathcal{F} = \bigcup \{X | \pi(X) \geq \pi\}.$$

An Example

A hypothetical transactional database is shown in Table 3.5. In the database, transaction IDs are displayed on each row and substructures are shown on each column. In each transaction, substructures A , B , C , D , E , and F can be present or absent. Note that these substructures can be subtree, subgraph, sublattices, etc. The presence of a substructure in a transaction is marked with a value of 1 , otherwise a value of 0 is shown.

Table 2.3: A transactional database.

ID	A	B	C	D	E	F
T_1	1	1	0	1	0	0
T_2	1	0	1	1	1	0
T_3	1	1	1	0	0	1
T_4	0	0	1	0	1	0

Table 2.4: Frequent substructures of the database shown in Table 3.5.

π^A	Frequent substructures
1	$A, B, C, D, E, F, AB, AC, AD, AE, AF, BC, BD, BF, CD, CE, CF, DE, ABC, ABD, ABF, ACD, ACF, ADE, ABCF, ACDE$
2	$A, B, C, D, E, F, AB, AC, AD$
3	A, C
4	\emptyset

From Table 3.5, the absolute support of substructure A , B , C , D , E , and F are 3, 2, 3, 2, 2, and 1, respectively. The support of substructure A is equal to $\pi^A(A) = \frac{3}{4} = 75\%$. Table 2.4 displays the frequent substructures discovered of different number of minimum supports from the above database.

2.3.3 Frequent Itemset Mining

Frequent itemset mining was illustrated in the market basket analysis (Agrawal and Srikant, 1994), where groups of items that shoppers often buy together are identified in order to boost sale.

In frequent itemset mining, the goal is to find sets of items that appear at least a user-specified number of times in a transactional database. The frequent itemset mining problem is a specific form of frequent pattern mining defined in section 2.3.2, with each set of items in the frequent itemset mining corresponds to a substructure in the frequent pattern mining.

Formal Definitions

Given a transactional database with $\mathcal{I} = \{i_1, i_2, \dots, i_m\}$ being the set of literals, called items. The database includes a set of n transactions, denoted by $T = \{T_1, T_2, \dots, T_n\}$, where each transaction T_i has a unique identifier, called its *tid*. T_i is a set of items such that $T_i \subseteq \mathcal{I}$.

A set of items (itemset) X occurs in transaction T_i , or T_i contains X , if $X \subseteq T_i$.

An itemset with k items is called a k -itemset. Y is a super-itemset of X if X is a proper subset of Y , that is, if $X \subset Y$.

Let $t(X) \subseteq T$ be the set consisting of all transactions that contain itemset X . The support of itemset X , $\pi^A(X)$, is the number of transactions in which X occurs as a subset. Therefore, $\pi^A(X) = |t(X)|$. Let the minimum support threshold be π , the frequent itemset mining problem is to find all itemsets (X) such that $\pi^A(X) \geq \pi$.

2.3.4 Frequent Subsequence Mining

A sequence is an ordered list of events (Han and Kamber, 2006). In this section, customer shopping sequence mining is discussed as an example of the general sequence mining. In shopping sequence data, each row is a chronically-ordered list of transactions belonged to a customer. Note that each transaction consists of a set of different items. Knowing sets of items are often purchased together by customers using frequent itemset mining is beneficial to a sales department. The department may want to identify sets of items that are often purchased in a particular order. For example, many customers are found buying certain products in a specific order such as ‘folate supplements, iron supplements, iodine supplements’, then ‘milk bottle, diaper’, and then ‘pram, toy’. Knowing these types of purchasing patterns is very useful for target marketing. This problem is known as frequent subsequence mining, or sequential mining (Pei et al., 2001; Agrawal and Srikant, 1995).

In general, given a sequence database with each transaction containing an ordered list of itemsets, the sequential mining problem is to find the subsequences of itemsets that appear at least a user-specified number of times. The frequent subsequence mining problem is a specific form of frequent pattern mining defined in Section 2.3.2, with each subsequence in the sequential mining corresponds to a substructure in the frequent pattern mining.

Formal Definitions

Given a transactional database with n sequences and $\mathcal{I} = \{i_1, i_2, \dots, i_m\}$ being a set of literals, called items. A sequence is an ordered list of itemsets, which is denoted by $\langle s_1 s_2 \dots s_l \rangle$, where s_j is an itemset and $s_j \subseteq \mathcal{I}$ for $1 \leq j \leq l$.

A sequence with length l is called an l -sequence. A sequence $\alpha = \langle a_1 a_2 \dots a_n \rangle$ is called a subsequence of sequence $\beta = \langle b_1 b_2 \dots b_m \rangle$, denoted by $\alpha \sqsubseteq \beta$, if there exist integers $1 \leq j_1 \leq j_2 \leq \dots \leq j_n \leq m$ such that $a_1 \subseteq b_{j_1}, a_2 \subseteq b_{j_2}, \dots, a_n \subseteq b_{j_n}$. If α is a subsequence of β , then β is the supersequence of α .

The database includes a set of n transactions, denoted by $T = \{T_1, T_2, \dots, T_n\}$, where each transaction T_i has a unique identifier, called its *tid* _{i} . T_i contains sequence X , if X is the subsequence of T_i , i.e. $X \sqsubseteq T_i$.

Let $t(X) \subseteq T$ be the set consisting of all transactions that contain sequence X . The support of sequence X , $\pi^A(X)$, is the number of transactions in which X occurs as a subsequence. Therefore, $\pi^A(X) = |t(X)|$. Let the minimum support threshold be π , the sequential mining problem is to find all subsequences (X) such that $\pi^A(X) \geq \pi$.

2.3.5 Frequent Subtree Mining

Frequent subtree mining problems will be discussed in detail in Section 2.4.3.

2.3.6 Frequent Subgraph Mining

The frequent subgraph mining field (Yan and Han, 2002; Kuramochi and Karypis, 2001; Jiang et al., 2013) has been studied extensively in the past. Although graph mining techniques can be applied to tree-structured data, it is not suitable for process logs due to its higher complexities compared to tree mining techniques.

2.3.7 Closed and Maximal Patterns

The pattern-based mining field often has to deal with a large number of frequent patterns (Han et al., 2002), which is hard to manage and understand. Although it is always possible to raise the support threshold to reduce the number of patterns, in reality, the threshold is often set low. This is because interesting, surprising patterns are expected to occur less frequently in many applications. However, setting a low minimum threshold affects the time performance of the algorithm and the number of patterns found. The lower the minimum support is, or the more complex the transactional database is (i.e. large number of instances, many frequent/recurring patterns, etc.), the more computational power or memory is required for a pattern-based mining algorithm. In the worst case scenario, the algorithm may not be able to terminate or no patterns can be found. Pattern compression is one possible solution to this problem.

The work in (Pasquier et al., 1999) suggested using closed or maximal itemsets to compress frequent itemsets. Broadly speaking, closed and maximal patterns are described as follows.

- Closed patterns: a closed pattern is a pattern that none of its super-patterns, e.g. super-itemset, super-sequence, have the same support. A closed frequent pattern is a pattern that is both closed and frequent.
- Maximal patterns: a maximal pattern is a pattern that none of its super-patterns, e.g. super-itemset, super-sequence, are frequent. A frequent maximal pattern is a pattern that is both maximal and frequent.

From a set of closed patterns discovered from the dataset, one can always identify the remaining frequent patterns and their supports. Therefore, the set of all closed patterns of a dataset is a lossless compression of the frequent patterns of that dataset. On the other hand, from maximal patterns, it is possible to identify the remaining frequent patterns but not their supports.

2.3.8 Frequent Pattern Mining Approaches

There are two main approaches in frequent pattern mining: *Apriori* (Agrawal and Srikant, 1994) and pattern growth (Han et al., 2000). These approaches are used in a wide range of frequent substructure mining. In order to explain the basic ideas of these approaches, frequent itemset mining is selected as an example.

Apriori algorithm (Agrawal and Srikant, 1994) is a level-wise method, where $(k-1)$ -itemsets are used to explore k -itemset. Initially, all frequent 1-itemsets in the transactional database are explored. On each level (k) , the algorithm performs the following two steps.

1. Generate candidates based on the frequent $(k-1)$ -itemsets. In this step, the algorithm joins $(k-1)$ -itemsets and prune those having at least one infrequent subset using the *Apriori* property ‘All non-empty subsets of a frequent itemset must also be frequent’ (Han and Kamber, 2006). This property belongs to a special category of properties called anti-monotonicity, which is understood as “if a set cannot pass a test, all of its supersets will fail the same test as well”.
2. Calculate the support for each candidate and eliminate those having supports lower than the minimum threshold.

The pattern growth approach (Han et al., 2000) adopts the *divide-and-conquer* strategy. Firstly, the database is compressed into a frequent pattern tree, or *FP-tree*, which keeps association information among items. Secondly, based on the *FP-tree*, the database is divided into conditional databases, each associated with one frequent item. Based on the conditional databases, frequent patterns are generated. A study on pattern growth performance shows that it is an order of magnitude faster than the *Apriori* algorithm (Han and Kamber, 2006).

Instead of representing data in a horizontal format, or *TID-itemset* format, where TID is a transaction ID, and itemset is a set of items for that transaction, the *ECLAT* algorithm (Zaki, 2000) represents data in a vertical format, or *item-TID_set* format, where TID_set is the set of transaction IDs containing the item. In this method, frequent k -itemsets are found from the intersection of $(k-1)$ -itemsets, and *Apriori* property is used to prune infrequent patterns. The advantage of using vertical data format is that there is no need to find the support of k -itemsets because it can be derived from the supports of $(k-1)$ -itemsets.

2.3.9 Association Rule Mining

In Section 1.3.3, an association rule is defined as $X \Rightarrow Y$, where X and Y are disjoint sets of items. However, the definition of X and Y can be extended to any type of substructures, e.g. subsequences, subtrees, subgraphs.

An association rule describes the association between two sets of substructures. It indicates the likelihood of the occurrence of a set of substructures (presented in the rule consequent (Y)) in a transaction, when a set of substructures (presented in the rule antecedent (X)) appear in that transaction.

Definition 2.4 (Rule support). *Given a transactional database of size n , the support of a rule $X \Rightarrow Y$ is defined as follows*

$$\pi(X \Rightarrow Y) = P(X \cup Y) = \frac{\pi^A(X \cup Y)}{n}.$$

Note that $P(X \cup Y)$ is the ratio of transactions in the database that contain both itemsets X and Y .

Definition 2.5 (Rule confidence). *A confidence of a rule $X \Rightarrow Y$ is defined as follows*

$$c(X \Rightarrow Y) = P(Y|X) = \frac{\pi^A(X \cup Y)}{\pi^A(X)}.$$

The association rule mining process comprises of two steps that are shown below:

1. Find all frequent itemsets having support larger than a user-specified threshold.
2. Generate association rules from the frequent itemsets. Prune rules that have confidence less than a user-specified threshold.

An often used example of association rule mining is the rule $\{\text{diapers}\} \Rightarrow \{\text{beers}\}$ found in a retail database. It shows that people who buy diapers are likely to buy beers at the same time (note that the opposite is not true). This hidden knowledge helps managers define their marketing strategies, for example, lowering the price of diapers and increasing the prices of beers to attract more customers and covering the losses for the diapers. It is also used for re-shelving purposes to boost sales.

2.3.10 Classification

The objective of classification is to extract models that describe important data classes. Such models, called classifiers, are used to predict categorical class labels (Han and Kamber, 2006). For example, historical weather data can be used to build a classifier that is able to estimate the chance of a storm coming in the next few days.

The first step in classification is the *learning* step, where a classifier, also known as prediction model, is built from training data. The training data is a set of instances associated with class labels, which are of nominal (or categorical) data type, and other attributes, which are called feature variables. The individual training instances' attributes are called predictor variables and the instances' class label attribute is called response variable. The prediction model can be in different representation such as decision trees, mathematical formula, or classification rules.

The second step in classification is *prediction*. If training data are used to evaluate the predictive accuracy of the classifier, the results would likely be optimistic (the classifier may learn some particular characteristics of the training data that do not appear in the general dataset) (Han and Kamber, 2006). Hence, test data should be used to estimate classifier's accuracy. The test data are made up of test instances and their associated class labels. The accuracy of a classifier is the percentage of test instances that are correctly classified.

Decision Tree Induction

In decision tree induction, a decision tree is learned from training data. A decision tree includes non-leaf nodes representing attributes, arcs, each of which connects two nodes and specifies a test on an attribute, and leaf nodes, which are class labels. An example of a decision tree classifier is shown in Fig. 2.3, which predicts three species of iris, e.g. *Setosa*, *Versicolor*, and *Virginica*. The feature variables include petal length and petal width. The two numbers that are shown next to the class label in each leaf node represent the number of items correctly classified and the total number of instances that satisfy the attribute tests along the path from root to that leaf node.

Given a test instance, for which the class label is unknown, attribute values are evaluated against conditions kept non-leaf nodes of the decision tree. This results in a path running from the root node to a leaf node which holds a class prediction for the instance.

Decision tree is one of the most common types of classifier. The strength of decision tree is that its algorithm does not require any domain knowledge, nor parameter settings. In addition, decision trees are intuitive for human and also support multidimensional data. Finally, decision tree induction is fast and its accuracy is comparatively-high in most cases.

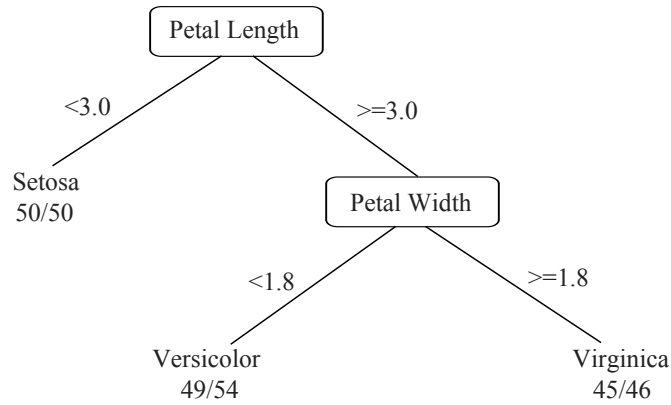


Figure 2.3: An example of a decision tree that predicts species of an iris.

The decision trees are built using a top-down and greedy (no backtracking) approach. On each level of a tree, an attribute selection measure such as information gain, gain ratio, *GINI* index, are used to identify an attribute that splits the training data into partitions that are as pure as possible. The purity of a group of instances is judged by the number of different class labels and the number of instances associated with each class label. Purity is maximised when all instances in a group are associated with a single class label. Each partition of data is then recursively examined to identify the remaining nodes of the decision tree.

The decision tree algorithm *Iterative Dichotomiser (ID3)*, which uses information gain as an attribute selection measure, was proposed in (Quinlan, 1986). Later, *C4.5* (Quinlan, 1993) was introduced as an upgrade of *ID3*, which uses gain ratio as an attribute selection measure. The *C4.5* algorithm is often used as a benchmark for other classification algorithms due to its speed and accuracy. A decision tree algorithm, *Classification and Regression Tree (CART)* (Wu et al., 2008), was developed concurrently with *C4.5*. This algorithm uses *GINI* index as an attribute selection measure and produces binary decision trees.

Rule-based Classification

A rule-based classifier uses a set of IF-THEN rules for classification, e.g. IF *condition* THEN *conclusion*. The “IF” part of a rule is the rule antecedent or precondition; the “THEN” part is the rule consequent. The rule antecedent contains one or more attribute tests that are logically *ANDed*. The rule consequent contains a class prediction.

If all attribute tests of a rule hold true for a given test instance, then it is said that the rule antecedent is satisfied by the instance, or the rule covers the instance, or the rule is triggered by the instance. In that case, the rule fires by returning the class prediction for the instance. In cases there are more than one rule satisfied, a conflict resolution strategy is needed.

In size ordering strategy, the triggering rule has the most attribute tests is fired. In

rule ordering strategy, the rules are sorted based on the most prevalent class, accuracy (see Definition 2.7), coverage (see Definition 2.6), etc. The first rule in the list that is triggered would fire its class prediction. If no rule is satisfied for a given instance, a default rule can be used to fire a default class, which is often the majority class in the training data, or the majority class of the instances that are not covered by any rule.

Let Y be the number of instances covered by a rule R ; X be the number of instances correctly classified by R ; and $|D|$ be the size of the database.

Definition 2.6 (Coverage). *The coverage of rule R is defined as follows*

$$CR(R) = \frac{Y}{|D|}.$$

Definition 2.7 (Accuracy). *The accuracy of rule R is defined as follows*

$$AR(R) = \frac{X}{Y}.$$

Classification rules can be derived from a decision tree, with each rule corresponds to a path from a root node to a leaf node. The decision tree shown in Fig. 2.3 corresponds to a set of rules shown below.

```
If Petal Length < 3.0 then iris_species = Setosa
If Petal Length ≥ 3.0 and Petal Width < 1.8
    then iris_species = Versicolor
If Petal Length ≥ 3.0 and Petal Width ≥ 1.8
    then iris_species = Virginica
```

One of the most widely used methods to learn classification rules is the sequential covering algorithm (Cohen, 1995; Han and Kamber, 2006). In this algorithm, rules are learned for one class at a time. Each rule for a given class should cover all (or many) of the training instances of that class and none (or few) of the instances of other classes. When a rule is learned, the training instances covered by the rule are removed. The process continues on the remaining instances until the terminating condition is met, e.g. the quality of a rule is below a user-specified threshold.

In the sequential covering algorithm, only the best rules, according to a rule quality measure, are selected in the rule learning process. Rule quality measures include accuracy, coverage, entropy (Quinlan, 1986), *FOIL*_gain (Quinlan, 1990). Entropy and *FOIL*_gain are used in the sequential covering algorithm *CN2* (Clark and Niblett, 1989) and *RIPPER* (Cohen, 1995), respectively.

2.3.11 Classification Evaluation

Examples of classification evaluation measures are accuracy, specificity, sensitivity, precision, F -measure, etc. Apart from being an evaluation measure, the word “accuracy” is used to denote a classifier’s predictive capability in general.

In the classification problem, if a test instance is the class of interest, it is called a *positive* instance. Otherwise, it is called a *negative* instance. A true positive (*TP*) instance is a positive instance that is predicted as positive. A true negative (*TN*) instance is a negative instance that is predicted as negative. A false positive (*FP*) instance is a negative instance that is predicted as positive. A false negative (*FN*) instance is a positive instance that is predicted as negative. The relations among *TP*, *TN*, *FP*, and *FN* are summarised in a two-dimensional confusion matrix (the number of dimensions depends on the number of class), which is shown in Table 2.5. *P* and *N* are the number of instances that are positive and negative, respectively. *P'* and *N'* are the number of instances that are predicted as positive and negative, respectively. The relations among *TP*, *TN*, *FP*, and *FN* are summarised in a two-dimensional confusion matrix (the number of dimensions depends on the number of class), which is shown in Table 2.5.

Table 2.5: A confusion matrix.

	Predicted class			
		Positive	Negative	Total
	Yes	TP	FN	P
	No	FP	TN	N
Actual class	Total	P'	N'	P+N

The accuracy of a classifier on a given test data is the percentage of test instances that are correctly classified.

Definition 2.8 (Accuracy). *The definition of accuracy is defined as:*

$$accuracy = \frac{TP + TN}{P + N}.$$

The misclassification (error) rate of a classifier on a given test data is the percentage of test instances that are incorrectly classified.

Definition 2.9 (Error rate). *The misclassification (error) rate is defined as:*

$$error\ rate = \frac{FP + FN}{P + N}.$$

The sensitivity measure, which is referred to as the *True Positive Rate (TPR)*, or recall, is the proportion of actual positive instances that are correctly classified as such. The sensitivity measure is used when people are interested in the accuracy of a classifier on the class of interest. For example, the performance of a cancer classification algorithm is based on the number of correct predictions on patients with a cancer, rather than the number of correct predictions on patients without a cancer.

Definition 2.10 (Sensitivity). *The sensitivity measure is computed as:*

$$sensitivity = \frac{TP}{P}.$$

The specificity measure is referred to as the *False Positive Rate (FPR)*, is the proportion of actual negative instances that are correctly classified as such.

Definition 2.11 (Specificity). *The specificity measure is computed as:*

$$specificity = \frac{TN}{N}.$$

The exactness of a classifier, the precision measure, is the proportion of predicted positive instances that are actually positive instances.

Definition 2.12 (Precision). *The precision measure is calculated as:*

$$precision = \frac{TP}{TP + FP}.$$

The completeness of a classifier, the recall measure, is the proportion of positive instances that are predicted as positive.

Definition 2.13 (Recall). *The recall measure is similar to the TPR (sensitivity) and calculated as:*

$$recall = \frac{TP}{TP + FN}.$$

Definition 2.14 (F-measure). *Since high precision and recall values are often achieved at the cost of the other, a harmonic measure that combines the two measures was defined as:*

$$F = \frac{2 \times precision \times recall}{precision + recall}.$$

2.3.12 Obtaining Reliable Accuracy Estimates

Depending on which data are used to train the classifier and which data are used to evaluate the classifier, the accuracy results can be varied. For this reason, to obtain a reliable estimate on the accuracy of the classifier, the following methods have been proposed.

Cross-validation

A common cross-validation method is *k-fold* cross-validation where the data are partitioned into *k* mutually exclusive subsets (folds) of equal size. The training and testing are done *k* times. In each iteration, a single partition is selected as the test set and the remaining partitions are used as training set. The accuracy estimate is the number of correct predictions from *k* iterations, divided by the total number of instances. A

common evaluation method is cross-validation when $k=2$, which is also called as hold-out method, where typically two-thirds of the original data are randomly selected for training purpose and the remaining data are used for testing purpose (Han and Kamber, 2006).

Bootstrap

In bootstrap methods, the training set are created by sampling the original data. One of the common sampling methods is the .632 bootstrap (Efron and Tibshirani, 1997). In each iteration, a data set with n instances is sampled (with replacement) n times and instances that are not sampled at all form the test set, whereas the remaining instances form the training set. After k iterations, the overall accuracy of the M is estimated as:

$$AR(M) = \frac{1}{k} \sum_{i=1}^k 0.632 \times AR(M_i)_{test_set} + 0.368 \times AR(M_i)_{train_set} \quad (2.1)$$

Note that $AR(M_i)_{test_set}$ is the accuracy of the model when applied to test set i and $AR(M_i)_{train_set}$ is the accuracy of the model when applied to the original data set.

Statistical Test

Although cross-validation provides a good estimate of the performance of a classifier, it might be that the results obtained are only due to chance. Statistical test can be used to ensure that one classifier's performance is statistically better than that of the other classifier. Some examples of statistical test used to evaluate a classification method include *Student's t-test*, two sample *t-test* (Han and Kamber, 2006).

ROC Curve

ROC analysis came from the signal detection theory (Egan, 1975). In *ROC* analysis, all test data are sorted based on their probability of being classified as a positive instance. In each iteration, a test instance is examined which follows the order of decreasing probability. A probability threshold is set equal to the probability of the examined test instance. Any instance whose probability is higher than the threshold is classified as positive; otherwise, it is classified as negative. A *TPR* and a *FPR* are calculated for the classifier. These numbers are then plotted on a graph where the vertical axis represents *TPR* values and the horizontal axis represents *FPR* values. The *ROC* curve connects pairs of *TPR* and *FPR* values for each test instance. The area under the *ROC* curve indicates the accuracy of the classifier.

2.3.13 Associative Classification

Associative classification (Thabtah, 2007; Liu et al., 1998; Yin and Han, 2003; Li et al., 2001) is an approach that uses frequent patterns to build prediction models. As mentioned in Section 2.3.9, frequent pattern mining is the first step in generating association rules. These rules illustrate the causal relationship among substructures in a transactional database. In classification, the need is to identify particular characteristics of training data that signal the occurrences of a certain class. Therefore, only association rules whose rule consequents are class labels are used for classification purpose. Classification approaches that are based on frequent patterns are called associative classification.

The three main steps of associative classification are as follows.

1. Frequent pattern mining: Discover frequent substructures in the training data set with a user-defined minimum support threshold.
2. Association rule generation and filtering: From frequent substructures, identify rules that have rule consequents as class labels. Rules having quality measure, such as confidence, lower than a user-specified threshold are filtered out. Note that, given an association rule $R: A \Rightarrow C$, the confidence value of R is the percentage of instances covered by A that have class label C . This measure is akin to rule accuracy (see Definition 2.7).
3. Evaluation: the set of rules learned from the previous steps form a rule-based model, which is then evaluated on a test data (see Section 2.3.10).

2.3.14 Interestingness

In association rule mining and associative classification, the number of discovered rules may be very large. However, only a portion of these rules are of interest to the user. Interestingness (Geng and Hamilton, 2006) is a broad concept that includes at least one of the following aspects.

- Conciseness: A pattern is concise if its structure contains a relatively few elements and a set of patterns is concise if it contains relatively few patterns;
- Coverage: A pattern has high coverage if it is present in a large subset of a data set;
- Reliability: A pattern is reliable if the relationship described by the pattern occurs in a high percentage of applicable cases;
- Peculiarity: A pattern is peculiar if it is far away from other discovered patterns according to some distance measure;

- Diversity: A pattern is diverse if its elements differ significantly from each other and a set of patterns is diverse if the patterns in the set differ significantly from each other;
- Novelty: A pattern is novel to a person if he or she did not know it before and is not able to infer it from other known patterns;
- Surprisingness: A pattern is surprising if it contradicts a person's existing knowledge or expectations;
- Utility: A pattern is of utility if its use by a person contributes to reaching a goal;
- Actionability: A pattern is actionable if it enables decision making about future actions in this domain.

Interestingness can be categorised into three main types, namely objective, subjective, and semantic interestingness. The objective interestingness is measured based on the probability and statistics applied to the data set. Objective interestingness represents the following aspects of interestingness: conciseness, generality, reliability, peculiarity and diversity.

In many cases, using objective interestingness only results in patterns that represent common sense. Thus, it is uninteresting. Subjective interestingness takes into account both the statistics and probability of the data and user requirements. Subjective interestingness represents the novelty and surprisingness aspects of interestingness. Semantic interestingness focuses on the explanations and semantics of the patterns discovered. Semantic interestingness represents the utility and actionability aspects of interestingness.

Two of the most popular objective interestingness measures used in association rule mining are *support* and *confidence* measure. These two measures are used to remove many weak or uninteresting rules from the rule sets. However, in many cases, rules that have high confidence value are still uninteresting (Han and Kamber, 2006). Thus, other objective interestingness measures such as lift or χ^2 might be used to further remove weak rules.

Definition 2.15 (Lift). *A lift measure of a rule $X \Rightarrow Y$ is defined as follows*

$$lift(X, Y) = \frac{P(X, Y)}{P(X) \times P(Y)} = \frac{c(X \Rightarrow Y)}{\pi(Y)}.$$

If the lift value is less than 1, then X is negatively correlated with Y , which means that the occurrence of one is likely to lead to the absence of the other. If the lift value is greater than 1, then X and Y are positively correlated, which means that the occurrence of one is likely to lead to the occurrence of the other. Otherwise, if the lift value is

1, then X and Y are independent, which means that the occurrence of one does not tell anything about the occurrence of the other. χ^2 is another way of identifying the relationship between two binary variables. Firstly, a contingency table is created from the data set with n instances, which is shown in Table 2.6. Each cell indicates the observed number of instances of the two variables.

Table 2.6: A contingency table for binary variables X and Y .

	Y	\bar{Y}	Total
X	o_{XY}	$o_{X\bar{Y}}$	o_X
\bar{X}	$o_{\bar{X}Y}$	$o_{\bar{X}\bar{Y}}$	$o_{\bar{X}}$
Total	o_Y	$o_{\bar{Y}}$	n

Secondly, the expected (the two variables are independent) number of instances of the two variables are calculated based on the following formula.

$$e_{ij} = \frac{o_i \times o_j}{N}. \quad (2.2)$$

Thirdly, the χ^2 is defined as.

$$\chi^2 = \sum \frac{(o_{ij} - e_{ij})^2}{e_{ij}}. \quad (2.3)$$

Consultation of the χ^2 distribution for 1-degree of freedom shows the probability of accepting the null hypothesis, which is X and Y are independent. A low probability value (compared to a statistical significance, e.g. 0.05) indicates there is a correlation between the two variables.

Apart from the objective interestingness measures discussed above, Table 2.7 lists some other popular measures (Geng and Hamilton, 2006).

2.4 XML Data Mining Methods

Since *MXML* and *XES* were endorsed as the two standards for logging event data, to the best of our knowledge there has been a lack of methods that are able to mine such documents. In this section, the key concepts and methods in *XML* mining are introduced. Section 2.4.1 shows that an *XML* document can be easily represented as a tree. Next, in Section 2.4.2, the formalism of tree-structured data is described. Frequent subtree mining is then discussed in Section 2.4.3. Finally, Section 2.5 introduces the *XML* classification problem.

Table 2.7: Objective interestingness measures.

Measure	Formula
Conviction	$\frac{P(X) \times P(\bar{Y})}{P(X\bar{Y})}$
Cosine	$\frac{P(XY)}{\sqrt{P(X) \times P(Y)}}$
GINI	$\max(P(X) \times (P(Y X)^2 + P(\bar{Y} X)^2) + P(\bar{X}) \times (P(Y \bar{X})^2 + P(\bar{Y} \bar{X})^2) - P(Y)^2 - P(\bar{Y})^2, P(Y) \times (P(X Y)^2 + P(\bar{X} Y)^2) + P(\bar{Y}) \times (P(X \bar{Y})^2 + P(\bar{X} \bar{Y})^2) - P(X)^2 - P(\bar{X})^2)$
Jaccard	$\frac{P(X,Y)}{P(X) + P(Y) - P(X,Y)}$
Laplace	$\max(\frac{N \times P(X,Y) + 1}{N \times P(X) + 2}, \frac{N \times P(X,Y) + 1}{N \times P(Y) + 2})$
Leverage	$P(Y X) - P(X) \times P(Y)$
Pearson	$\frac{P(X,Y) - P(X) \times P(Y)}{P(X) \times P(Y) \times (1 - P(X)) \times (1 - P(Y))}$
Yule's Q	$\frac{P(X,Y) \times P(\bar{X}, \bar{Y}) - P(X, \bar{Y}) \times P(\bar{X}, Y)}{P(X,Y) \times P(\bar{X}, \bar{Y}) + P(X, \bar{Y}) \times P(\bar{X}, Y)}$
Lift	$\frac{P(X Y)}{P(X)}$ or $\frac{P(X,Y)}{P(X) \times P(Y)}$

2.4.1 The Relation between XML and Tree-structured Data

As mentioned in Chapter 1, an XML document can be mined from its content, structure or both content and structure. Event logs are highly-structured while having low textual contents. Elements in an event log such as activities, timestamps, and events have well-defined meanings and their positions in a document can be located using data schemas. To exploit these properties of process logs, this thesis focuses on the problem of mining structural patterns of the data.

It is possible to convert an XML-based process log to a tree database, as seen in Listing 1.1 and Fig. 1.5, due to the strong resemblance between XML and tree-structured data. In XML, the elements (tags) are organised in a hierarchical way, i.e. each element may have one or more child elements, an element value, and one parent element, except for the root element ($\langle \text{xml} \rangle$), which has no parent. An element may have one or more attributes, each of which has a value. Elements, element data, attributes, attribute values and the relations among them correspond to nodes and edges in a tree-structured data.

In more detail, the root element of an XML document, i.e. $\langle \text{xml} \rangle$ corresponds to the root node of a rooted tree (A rooted tree is a tree whose one of its nodes is distinguished from others, which is called the root). Children of an XML element are positioned in a certain order, which corresponds to an ordered tree. If a node has k

children, the child at the left-most position is at position 0, and children to its right have incrementing position order, up to $k-1$. Since each item of an XML document has a name or value, the whole document corresponds to a rooted labelled ordered tree. A framework for mapping XML data into tree structures and makes them ready for association rule algorithm was given in (Feng et al., 2003).

It was shown that an XML document can be straightforwardly represented by a rooted labelled ordered tree. From this point forward, our discussion is focused on rooted labelled ordered tree data mining.

2.4.2 Formalism of Tree-structured Data

In this thesis, tree-structured databases that consists of a set of rooted ordered labelled trees are investigated. The notation of tree-structured data will be developed based on the concepts of graph-based data (Chi et al., 2004a).

A labelled graph is denoted as $G=(V,E,\xi,L)$, where V is a set of vertices, $E = \{(v_1,v_2)|v_1,v_2 \in V \text{ AND } v_1 \neq v_2\}$ is a set of edges, ξ is an alphabet of vertex and edge labels, and $L : V \cup E \rightarrow \xi$ is a labelling function that assign labels to vertices and edges.

A directed graph is a graph whose edges are ordered pairs of vertices. A undirected graph is a graph whose edges are unordered pair of vertices.

A path is a list of vertices of a graph such that each pair of neighbouring vertices in the list is an edge of the graph (Chi et al., 2004a). The length of a path is defined by the number of edges in the path. A cycle is a path of which the first and the last vertices are the same.

An acyclic graph is a graph that has no cycle. An undirected graph is connected if there is at least one path between any pair of vertices, whereas an undirected graph is disconnected if there exists a pair of vertices such that no path is found between them.

A free tree is a graph that is undirected, connected and acyclic. A rooted unordered tree is a undirected, connected, acyclic graph with one distinguished vertex called the root. There is a unique path from the root to every other vertex. If vertex x is on the path from the root to vertex y , then x is an ancestor of y (and y is a descendant of x), denoted as $x \leq_p y$, where p is the length of the path from x to y . In case $(x,y) \in E$, x is the parent of y (and y is a child of x), which is denoted as $x \leq_1 y$. If vertex y and z are both children of vertex x , then y and z are siblings. In case two vertices share the same ancestor, they are called cousins. A rooted ordered tree is a rooted tree that has a predefined total ordering (\ll) among each set of siblings. When a rooted ordered tree is presented graphically, the order between siblings is implied by the left-to-right order. The expression $x \ll y$ is used to describe node x and node y are siblings, with node x is to the left of node y . If node x and node y are immediate siblings, meaning that node x locates to the left of node y , and there are no other nodes in between the two nodes,

such relations between x and y is denoted as $x \ll_i y$. If a vertex has k children, then its leftmost child is the first child, its next child is the second child, etc., and its last child is the k th child.

The fan-out (degree) of a vertex in a tree is the number of children whose parent is that vertex. The average (max) fan-out of a tree is the average (max) fan-out of all vertices in that tree. The level/depth of a vertex is the length of the path from root to that vertex. The height of a tree is the number of vertices in the longest path starting from the root. The size of a tree, denoted as $|T|$, is the number of vertices the tree has. A forest is a set of one or more disjoint trees.

Definition 2.16 (Isomorphic subtree). *Given a tree $S = (V_S, E_S, L_S)$ and tree $T = (V_T, E_T, L_T)$, S is an isomorphic subtree of T , denoted as $S \preceq T$, if and only if there exists an injective function $f: V_S \rightarrow V_T$, such that $(x, y) \in E_S$ if and only if $(f(x), f(y)) \in E_T$.*

Note that isomorphic subtree relation is also called tree inclusion relation. If f is a bijective function, then S and T are called isomorphic.

Definition 2.17 (Induced subtree). *Given a tree $S = (V_S, E_S, L_S)$ and tree $T = (V_T, E_T, L_T)$, tree S is an induced subtree of T , denoted as $S \preceq_i T$, if and only if there exists an injective function $f: V_S \rightarrow V_T$, such that:*

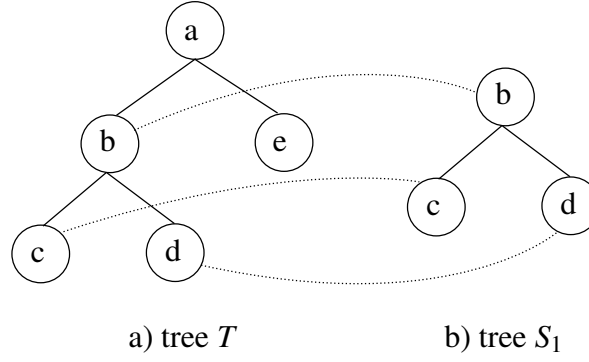
- $(x, y) \in E_S$ if and only if $(f(x), f(y)) \in E_T$ (S is an isomorphic subtree of T);
- $x \ll y$, $x, y \in V_S$ (x and y are siblings and x is on the left of y) if and only if $f(x) \ll f(y)$;
- $L_S(x) = L_T(f(x))$, $\forall x \in V_S$.

In other words, f preserves the parent-child relationships, the siblings order, as well as vertex labels.

In Fig. 2.4, each vertex in tree S_1 is mapped into a vertex in tree T . In addition, the parent-child relationships and vertex labels are preserved by the mapping. Therefore, S_1 is an induced subtree of T .

Definition 2.18 (Embedded subtree). *Given a tree $S = (V_S, E_S, L_S)$ and tree $T = (V_T, E_T, L_T)$, tree S is an embedded subtree of T , denoted as $S \preceq_e T$, if and only if there exists an injective function $f: V_S \rightarrow V_T$, such that:*

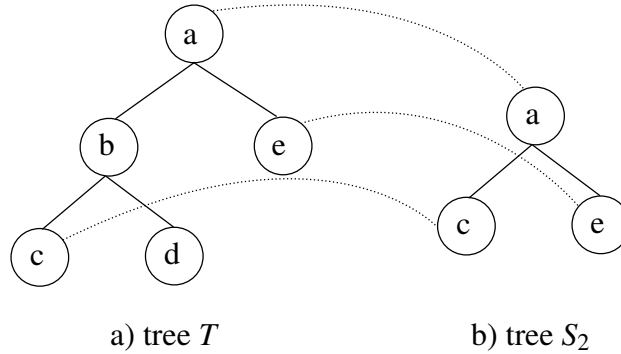
- $(x, y) \in E_S$, $x \leq_1 y$ if and only if $f(x) \leq_p f(y)$, $p \in \mathbb{N}$, $p > 0$;
- $x \ll y$, $x, y \in V_S$ (x and y are siblings and x is on the left of y) if and only if $\exists m, \exists n \in V_T : m \leq_p f(x), n \leq_q f(y), m \ll n, p \in \mathbb{N}, q \in \mathbb{N}, p > 0, q > 0$;

Figure 2.4: S_1 is an induced subtree of T .

- $L_S(x) = L_T(f(x)), \forall x \in V_S$.

In other words, f preserves the ancestor-descendant relationships (a parent of a node in an embedded subtree is an ancestor of that node in the original tree), as well as vertex labels.

Embedded subtrees are a generalisation of induced subtrees. In Fig. 2.5, each vertex in tree S_1 is mapped into a vertex in tree T . In addition, the ancestor-descendant relationships and vertex labels are preserved by the mapping. Thus, S_2 is an embedded subtree of T .

Figure 2.5: S_2 is an embedded subtree of T .

Tree S is said to occur in tree T , or T contains S , if S is an induced or embedded subtree in T , i.e. $S \preceq_e T$ or $S \preceq_i T$. S can occur multiple times in T and each occurrence is identified by a sequence $f(x_0)f(x_1) \cdots f(x_{|S|})$, where $x_i \in V_S$.

Let $p(S, T)$ be an occurrence of S in tree T : $p(S, T) = \{f(v) | \forall v \in V_S\}$. The number of occurrences of S in T is denoted as $n_T(S)$, where $n_T(S) = |\{p(f, S, T) | f: V_S \rightarrow V_T, \text{ such that } S \prec_e T\}|$.

Let $d_T(S)$ represent the occurrence indicator of S in T , with $d_T(S) = 1$ if $n_T(S) > 0$ and $d_T(S) = 0$ if $n_T(S) = 0$. In other words, $d_T(S) = 1$ if and only if $S \prec_i T$ or $S \prec_e T$.

Let DB be a database (forest) of trees. The support of subtree S in DB is defined as $\pi^A(S) = |d_T(S)|, \forall T \in DB$. That is, the support of subtree S in DB is the number of trees in DB that contain at least one occurrence of S .

The weighted support of S in DB is defined as the total number of occurrences of S over DB , i.e. $\pi^W(S) = \sum_{T \in DB} n_T(S)$. The weighted support can also be defined as $\pi^W(S) = \frac{\sum_{T \in DB} n_T(S)}{\sum_{T \in DB} 1}$.

A subtree is frequent if its support is more than or equal to a user-defined minimum support threshold.

Representation of Trees

Tree is a fundamental data structure which can be represented in a number of ways. In the computer memory, a tree can be stored as a set of vertices which are linked by pointers. Adjacency matrix is a common way of representing a tree (and graph). In the adjacency matrix, each row and column indicates a vertex. A cell in a table has a value of 1 if there is an edge between the two vertices represented by the corresponding row and column of the cell; the cell has a value of 0, otherwise.

There are many other ways of representing trees, such as adjacency list, breadth-first, depth-first canonical form (Chi et al., 2005b). In (Zaki, 2002), the author claimed that using string to encode trees is a space-efficient method. A pre-order string encoding of a tree T is denoted as $\phi(T)$.

Initially, $\phi(T) = \emptyset$. A depth-first pre-order traversal is performed on the tree, starting from the root. Whenever a vertex is reached, its label is added to $\phi(T)$. In case a backtrack is performed, a symbol -1 is added to $\phi(T)$. For example, the pre-order string encoding of tree S_1 in Fig. 2.4 is $b c -1 d -1$. That is, the method starts from the root of S_1 and add b to the string. It then traverses to the next vertex, c , which is then appended to the encoding. Since there is no children of c , the algorithm backtracks to a and add -1 to the string. Next, vertex d is traversed, and d is added to the encoding. Finally, the method backtracks to the root, adding -1 to $\phi(T)$.

A vertex can be referred to by its order in the pre-order traversal of the tree. The root is visited first, thus it has a position of X_0 ; the first child of the root (its left-most child) is visited next, thus its position is X_1 . In this fashion, all other vertices are visited and labelled; the position of last vertex that is visited is X_{n-1} , where n is the number of vertices. Fig. 2.6 shows an example of a tree where its vertices are labelled according to their position in the pre-order traversal.

In Fig. 2.5, the subtree S_2 occurs in tree T at position X_0 , X_2 , and X_4 . These locations can be written as 024.

2.4.3 Frequent Subtree Mining

Frequent subtree mining is a problem of finding all subtrees of a tree database that occur at least a number of times that is greater than the minimum support threshold. It

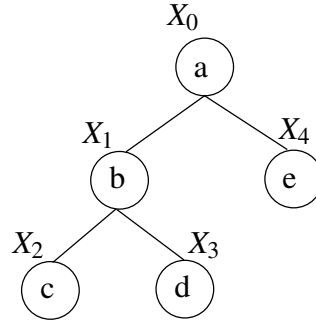


Figure 2.6: Position of vertices in a tree.

is a specific form of frequent pattern mining defined in Section 2.3.2, with each subtree in frequent subtree mining corresponds to a substructure in frequent pattern mining.

Frequent subtree mining has applications in different domains, e.g. Web log analysis (Zaki, 2005b; Zaki and Aggarwal, 2006; van der Aalst and van Dongen, 2002; Zaki, 2005a, 2002; Chi et al., 2005b; Tan et al., 2006; Hadzic, 2012), Internet routing (Chi et al., 2003), chemical compound analysis (Chi et al., 2003; Rückert and Kramer, 2004; Chi et al., 2004b; Deshpande et al., 2005; Dehaspe et al., 1998), bioinformatics (Borgelt and Berthold, 2002; Zhang and Wang, 2008; Deepak et al., 2014), structure discovery (Wang and Liu, 1998).

Definition 2.19 (Frequent subtree mining (Zaki, 2002)). *Given a tree database, a minimum support threshold π , and a isomorphic subtree relation \preceq , the set of frequent subtrees in the database (\mathcal{F}) is defined as:*

$$\mathcal{F}(\pi, \preceq) = \bigcup \{S \mid \pi(S) \geq \pi\}.$$

Given S_1 is a subtree of S_2 and a tree database, there is an anti-mononicity relation between the supports of S_1 and S_2 , which is given as follows.

$$S_1 \preceq S_2 \Rightarrow \pi(S_1) \geq \pi(S_2). \quad (2.4)$$

2.4.4 Frequent Subtree Mining Algorithms

A categorisation of common frequent subtree mining algorithms is given in Table 2.8. The categorisation is based on: (i) types of input trees, e.g. ordered, unordered, free, hybrid, (ii) types of tree inclusion relations, e.g. induced, embedded, and (iii) types of condensed representation, e.g. maximal, closed. Note that all free trees are unordered trees. Frequent hybrid subtree mining includes algorithms that are able to mine a combination of different input trees. For example, *POTMiner* (Jiménez et al., 2010) mines ordered, unordered, and partially-ordered subtrees. *TRIPS* (Tatikonda et al., 2006) and *TIDES* (Tatikonda et al., 2006) are able to mine ordered, unordered, labelled or

unlabelled, edge-labelled subtrees. *CMTreeMiner* (Chi et al., 2005a) can mine both ordered, unordered maximal and closed induced subtrees. *DryadeParent* (Termier et al., 2008) mines a special type of tree that is called attribute trees (In attribute trees, all siblings must have different labels). The algorithm *Phylominer* (Zhang and Wang, 2008) can mine both free and rooted trees. An idea of mining patterns in carbohydrate sugar chains using α -closed subtrees was proposed in (Hashimoto et al., 2008). The user-specified parameter α is used to reduce the number of closed subtrees found in the dataset. Recently, *BOSTER* (Chowdhury and Nayak, 2013, 2014) was proposed to mine frequent induced, unordered subtrees.

Table 2.8: Common frequent subtree mining algorithms.

	Maximal	Closed	Induced	Embedded
Unordered tree mining				
TreeFinder (Termier et al., 2002)	*			*
uFreqT (Nijssen and Kok, 2003)			*	
CousinPair (Shasha et al., 2004)				*
RootedTreeMiner (Chi et al., 2005b)			*	
SLEUTH (Zaki, 2005a)				*
Unot (Asai et al., 2003)				*
Uni3 (Hadzic et al., 2007)			*	
PathJoin (Xiao et al., 2003)	*		*	
BOSTER (Chowdhury and Nayak, 2014)			*	
Ordered tree mining				
FREQT (Asai et al., 2002)			*	
TreeMiner (Zaki, 2005b)				*
Chopper (Wang et al., 2004)				*
Xspanner (Wang et al., 2004)				*
AMIOT (Hido and Kawano, 2005)			*	
MB3-Miner (Tan et al., 2005)				*
IMB3-Miner (Tan et al., 2006)			*	*
Free tree mining				
FreeTreeMiner (Chi et al., 2003)			*	
FreeTreeMiner (Rückert and Kramer, 2004)			*	
HybridTreeMiner (Chi et al., 2004b)			*	
GASTON (Nijssen and Kok, 2004)			*	
Phylominer (Zhang and Wang, 2008)			*	*
Hybrid tree mining				
CMTreeMiner (Chi et al., 2005a)	*	*	*	
POTMiner (Jiménez et al., 2010)			*	*
TRIPS (Tatikonda et al., 2006)			*	*
TIDES (Tatikonda et al., 2006)			*	*
HybridTreeMiner			*	

The common approach in frequent subtree mining is firstly generating candidates and secondly counting the support of each candidate. There are several methods for

the candidate generation process such as right most path extension (used by *Freqt*, *Unot*, *uFreqt*, *GASTON*, *TIDES*), equivalence-class-based extension (used by *TreeMiner*, *SLEUTH*, *Phylominer*, *POTMiner*, *MB3Miner*, *IMB3Miner*), extension and join (used by *HybridTreeMiner*), right and left tree join (used by *AMIOT*), *Apriori* itemset generation (*TreeFinder*), enumeration tree (used by *CMTreeMiner*, *RootedTreeMiner*), self-join (used by *FreeTreeMiner*) and cousin distance (use by *CousinPair*). Methods that are based on pattern growth approach are *PathJoin*, *Chopper* and *XSpanner*. The first method uses compacted structures called *FP-Trees* to compress input data, while the last two encodes trees in sequence format.

The main approaches for support counting includes scope list, which is used in *TreeMiner*, *SLEUTH*, *POTMiner*, and occurrence list, which is used in *Freqt*, *AMIOT*, *HybridTreeMiner*, *MB3Miner*, *IMB3Miner*, *Unot*, *RootedTreeMiner*. Other methods are bipartite graphs (*uFreqT*), clustering techniques (*TreeFinder*), hash table (*TRIPS*, *TIDES*).

2.5 Structural XML Classification

As mentioned in Chapter 1, classification algorithms are used in many process mining problems such as root cause analysis, time prediction, activity recommendation, etc. In this thesis, the focus is on *inter-structure mining*, where the structural properties of XML documents are examined instead of their textual/semantic content. Notable *inter-structure XML mining* studies are described in (Zaki and Aggarwal, 2006; Costa et al., 2013b; Bringmann and Zimmermann, 2005; De Knijf, 2007; Garboni et al., 2006; Chehreghani et al., 2009; Geamsakul et al., 2005; Cheng et al., 2008; Kim et al., 2010). Examples of studies that use the latter approaches include (Wang et al., 2012; Costa et al., 2011; Denoyer and Gallinari, 2004). As mentioned in Section 2.4.1, the scope of this thesis is limited to mining only structural properties of XML documents.

Many tree-structured classification algorithms were inspired by the associative classification framework for flat data (also called *Classification Based on Association* or *CBA*) first introduced in (Liu et al., 1998). In this framework, rules are generated from the training data (Agrawal and Srikant, 1994) and only rules having rule consequents as class labels are selected as part of the rule set; rules having less predictive power or being not statistically significant are removed.

Later, the method *CMAR* (Li et al., 2001) improved *CBA* by using an extended version of *FP-Growth* (Han et al., 2000) for mining rules. Multiple rules are used for predicting class instead of one as in *CBA*. Moreover, χ^2 value is used to prune rules that are not statistically significant.

A hybrid method, *CPAR* (Yin and Han, 2003), was proposed based on the support confidence framework and *FOIL* (First Order Inductive Logic Programming) (Quinlan

and Cameron-Jones, 1993), which is a greedy approach that learns rules to distinguish positive examples from negative ones. The decay factor and multiple laterals generation were proposed to reduce the missing of important rules. In *CPAR*, *Laplace* accuracy is used as a rule strength measure and only top- k rules are used for prediction. Other methods extending the associative classification framework can be found in (Veloso et al., 2006; Dong et al., 1999; Wang et al., 2000; Thabtah et al., 2004). A complete survey of the associative classification framework can be found in (Thabtah, 2007).

One of the first structural classifiers that is based on the associative framework is *XRules* (Zaki and Aggarwal, 2006). In this method, frequent embedded subtrees are mined using *XMiner* (an extension of *TreeMiner* (Zaki, 2005b)) that can simultaneously mine all frequent subtrees related to each class. The associations among the subtrees and their classes form the rule set. After that, the rule strength of each rule for each class is calculated based on different measures such as confidence, likelihood or weighted confidence. Finally, each test instances is matched against rules in the rule set and the combined rule strengths of the triggered rules are compared between classes. Class that has higher combined rule strength than others' is selected as a label for the test instance.

A tree-structured associative classifier, called *X-Class* (Costa et al., 2013b), uses four different types of substructure, e.g. distinct nodes, distinct edges, root-to-node paths and root-to-leaf paths as patterns. It is claimed that these substructures are better than subtree in terms of discriminative power and interpretability. Among the four, the substructure root-to-leaf path is the most effective and gives compact rule sets.

In *Tree²* (Bringmann and Zimmermann, 2005), statistical measures such as χ^2 and *Information Gain* are used to select the most discriminative subtree patterns for the decision tree algorithm. The convex property of these measures is used in the branch-and-bound search for subtree patterns. Despite not having the best accuracy results, the number of generated patterns is small and easy to understand. Decision tree Graph-Based Induction (Geamsakul et al., 2005) is a similar method which uses graph patterns instead of trees.

In (De Knijf, 2007), the *FAT-CAT* method is based on attribute trees (Arimura and Uno, 2005), where in emerging subtrees are identified by combining a local and a global frequent pattern mining. The subtrees are then used as binary features for the decision tree algorithm. Note that a subtree is emerging if it frequently occurs in one class and rarely with other classes.

Both *Tree²* and *FAT-CAT* methods are based on decision tree learning. The former method builds decision tree step-by-step and the splitting attributes, i.e. discriminative subtrees, are discovered on the fly. The latter method emphasizes on the discovering of highly frequent and discriminative attribute subtree and use them as binary features

while leave the choice of a decision tree algorithm for the user.

Another tree-structured classification approach is to directly mine discriminative subtrees that are used as the conditions of classification rules. This idea was first implemented the *DDPMine* (Cheng et al., 2008) algorithm for itemset data. In this work, the *FP-Growth* algorithm (Han et al., 2000) was utilized to generate frequent itemsets and to perform the rule ranking at the same time. Statistical measures that describe the discriminative power of a pattern such as *Information Gain* are used to select the best rules. The upper bound of the discriminative measures is used to skip the conditional *FP-Tree*, thereby reduces the pattern search space.

NDPMine (Kim et al., 2010) is a pattern-based classification method that was inspired from *DDPMine* but uses the number of patterns as features for *SVM* algorithm. This method utilizes a shrinking memory algorithm for better performance. It was used in conjunction with *CMTreeMiner* for the classification of authorship data that is represented in tree-structured format.

The tree-structured classification problem can also be solved by transforming trees into sets of attributes. Selecting a right attribute plays an important role for the efficacy of these methods. In (Candillier et al., 2006), trees are transformed into sets of attribute-values, which are then mined by various existing methods of classification and clustering. The selected attributes represent different types of structural properties, such as parent-child, next-sibling, and path-from-root relations.

In (Garboni et al., 2006), *XML* documents are represented in a form of node sequences where each node is attached with its corresponding level in the original tree. Sequential pattern mining algorithms are used to extract sets of frequent sequences which are representative of different classes and then matched against the test data.

Another class of methods that can be used for *XML* document classification is based on kernel methods. Some of the studies using kernels for tree-structured data are presented in (Vishwanathan and Smola, 2004; Kashima and Koyanagi, 2002; Aioli et al., 2009). Although kernel methods are powerful machine learning tools that can be used in a variety of tasks, such as classification, clustering, creating a kernel function that is best suited to a specific application is still a difficult problem (Costa et al., 2013b).

There are many *XML*-based/tree-structured classification methods which can be used on process logs. However, to the best of our knowledge, none of them take into accounts the exact order of each event and/or its attributes in the process of constructing a prediction model. The position information of subtree patterns were known to be important in process mining, as will be described in the motivating example in Chapter 3. Therefore, it is of interest to build a classifier for event logs which considers the position of subtree.

2.6 Summary and Research Gaps

This chapter reviewed the necessary background of the *XML*-based event log mining problem. Section 2.1 gave an overview of the application context—the *BPM* field. After that, the process mining problem was discussed in Section 2.2. The structure of an event log—the starting point of every process mining task—was also presented. Data mining techniques that are commonly used in process mining such as frequent pattern (subsequence) mining, association rule mining, and classification were introduced in Section 2.3. Classification evaluation methods was introduced for use in experiments. Associative classifier, one of the major method in classification, was presented next. Data mining plays a major part of process mining. However, it has not been fully utilised in process mining because most data mining techniques are used for the purpose of discovering process models from event data. This raises the need for more studies on process mining methods that are independent of process models.

Since the *XES* standard was adopted by the IEEE Task Force on Process Mining (ICTFPM, 2010), there has been a need to develop methods that can directly mine *XML*-based process logs. *XML* mining methods has been extensively studied in many areas, such as biology and Web usage mining. Such methods can certainly be applied to *XES*-based event logs, however due to the complexities of event logs (commonly having a large number of event nodes), more efficient mining methods are always in need. There are two approaches in mining *XML* documents: (i) examine the connections between *XML* entities (e.g. tag), and (ii) examine the connections and the semantics/content of the entities. This thesis focuses on former approach. Background on representing *XML* data in tree-structured format and tree data structure formalism were presented in Section 2.4.

Tree-structured data mining is another established research area. One of the most important parts of tree mining is frequent subtree mining, which was described in Section 2.4.3. Section 2.5 reported different classification methods on tree-structured data. It is expected that those methods could work on *XML*-based process logs; however, none of them are able to show the exact order of a tree pattern, which is in many cases important in the context of process mining (Bui et al., 2012b, 2015). In addition, the study in (Hadzic et al., 2015) shows that methods that are based on frequent subtree mining could have performance issues when mining process logs at lower support thresholds. The above-mentioned motivated us to develop a method that can mine semi-structured process logs using position-constrained subtree mining, which will be described in Chapter 3. This method can be easily adjusted to utilise traditional frequent subtree mining techniques for process mining purpose. Moreover, this chapter introduced an exploratory process log analysis technique that is based on the proposed method. Chapter 4 focused on the evaluation of the proposed process mining method

and its derived exploratory analysis method on the practical process mining tasks such as prediction an outcome of a running process instance and recommending actions that could lead to a desired business goal. Chapter 5 introduced an adaptation of associative classification to our method and evaluated its application not only in event logs but also datasets from other domains.

Chapter 3

PCFSM: A New Method for Mining Process Logs

Semi-structured data such as *XML* documents are known for their ability to represent the contextual information among different data items in a domain-specific way. Recently, *XML*-based languages have been increasingly used to represent event logs, especially after the proposal of *MXML* and *XES*. Other attempts at using *XML* to store event logs were presented in (Kim, 2006; Gonçalves et al., 2002). As mentioned earlier, *XML* documents can be represented as rooted ordered labelled trees, due to their hierarchical nature. This encourages the utilisation of tree-structured data mining methods on *XML*-based process logs.

Other than our preliminary work proposed in (Bui et al., 2012b,a, 2015), to the best of our knowledge, no tree-structured data mining techniques have been specifically explored for the purpose of mining *MXML/XES* event logs. Frequent subtree mining has been the basis for discovering interesting associations among tree-structured data objects in *XML* data (Feng et al., 2003; Mazuran et al., 2009b; Moradi and Keyvanpour, 2015), but their utilisation in the process mining field is still to be explored. For example, a (synthetic) process log dataset was used in the work of Hadzic et al. (2011a), but the main purpose of this work is to compare the time performance and quality of clustering solution between algorithms. The study in (Greco et al., 2005) demonstrated the use frequent pattern mining to discover workflow and weak patterns in event data; however, this method is not designed specifically for *XML*-based process logs. The same holds for other data mining methods that take structural aspects into account such as *XML* clustering (Kutty et al., 2011; Hadzic et al., 2011a) and classification (Kim et al., 2010).

The process mining field would be benefited by the application of tree-structured data mining techniques, as they not only preserve the order of events, but also their context and structural organisation within the process. In the case of web logs, it has already been shown in (Hadzic et al., 2011b; Zaki, 2005b) that subtree patterns are

more informative than itemsets or sequential patterns as they capture the structural properties as well as the navigational behaviour over the web site structure. Hence, in the case of process logs, a similar reasoning would apply, and subtree patterns would capture the execution patterns over the entire structure of the business process at hand.

Section 3.1 presents a motivating example for using tree mining techniques to mine process logs. In addition, from the example, the need for a more expressive representation of the process log is highlighted. The structure-preserving tree mining approach is described and formalised in Section 3.2. The time performance of this method is presented in Section 3.3. After that, a process log analysis method, which is based on the position-constrained frequent subtree mining method, is proposed in Section 3.4. Finally, a discussion on the proposed method is given in Section 3.5.

3.1 Mining Process Logs Using Tree-based Techniques

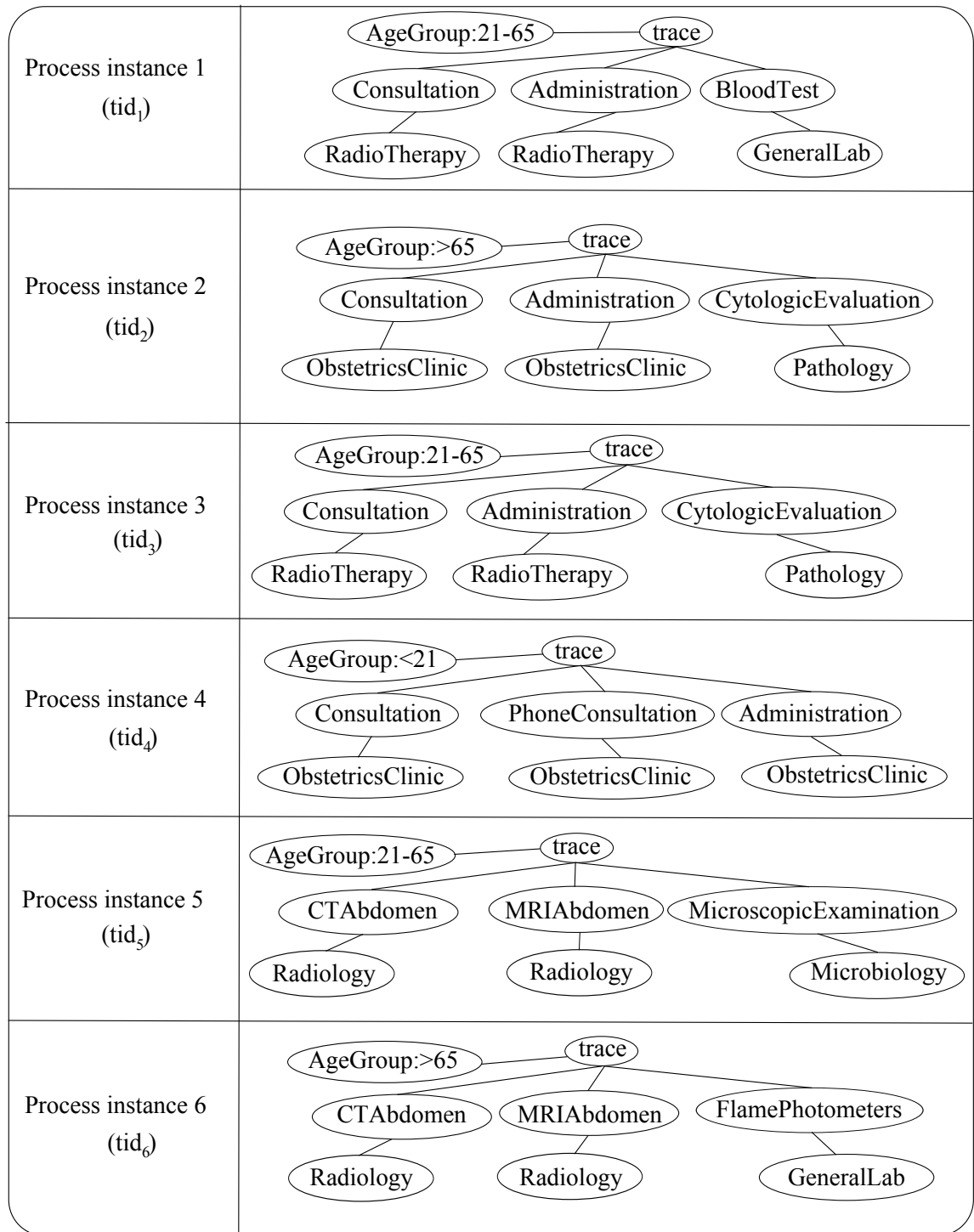
This section is organised into two parts. Firstly, a motivating example for using tree-based techniques to mine process logs instead of traditional methods is presented. Secondly, the time performance of the position-constrained frequent subtree mining method is compared to traditional approaches.

3.1.1 A Motivating Example

In this section, the example of the hospital process log described in Chapter 1 is revisited. The tree-structured database representing the process log is shown in Fig. 3.1.

As mentioned earlier, most current process mining methods focus on the control-flow perspective of a process and often ignore the data attributes that come together with the events. Although the data presented in Fig. 3.1 are rich in content, process mining algorithms such as α -algorithm (Van der Aalst et al., 2004), Heuristic Miners (Weijters and van der Aalst, 2003), Fuzzy Miner (Günther and van Der Aalst, 2007) only understand them as sets of sequences of events, and from that discover the generative process model. An example of a process model discovered by a heuristic algorithm is shown in Fig. 3.2. It is noticeable that the process model does not include any information about the attribute values of each event such as the departments where the activities were administered.

In order to incorporate the contextual information of each event into the mining process, the set of traces are considered as a set of trees rather than as a set of sequences as is commonly done. Using the frequent closed subtree mining (Chi et al., 2005a) on the tree database gives us a number of subtree patterns, each of which occurs at different traces. An example of closed subtrees at support = 2 is given Fig. 3.3.

Figure 3.1: T_{HOS} —the hospital database (reproduced from Chapter 1 for convenience).

The subtree pattern in Fig. 3.3(a) occurs in process instance 1 and 3 of Fig. 3.1. It is seen that the pattern preserves the order of events (according to pre-order traversal of the subtree) and thus represents the frequent path of executions. The path *Consultation* \Rightarrow *Administration* does not contain the event *PhoneConsultation*, *BloodTest* and *CytologicEvaluation* because the minimum support value is set to 2; if that parameter is lowered, longer paths of executions can be detected. The benefit of using subtree mining is confirmed here where activities and their contextual information are preserved in the pattern, and the path of execution information is enriched with their administering departments, e.g. *RadioTherapy*, *ObstetricsClinic*.

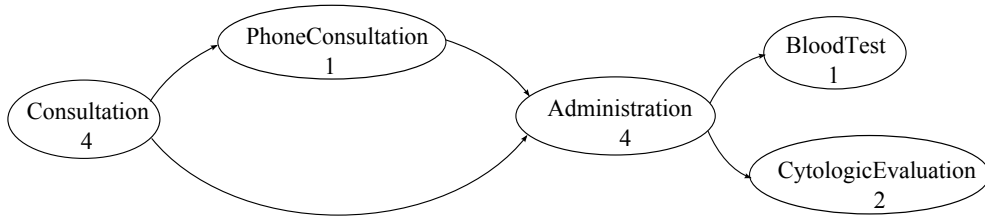


Figure 3.2: Patterns learned from Heuristic Miner.

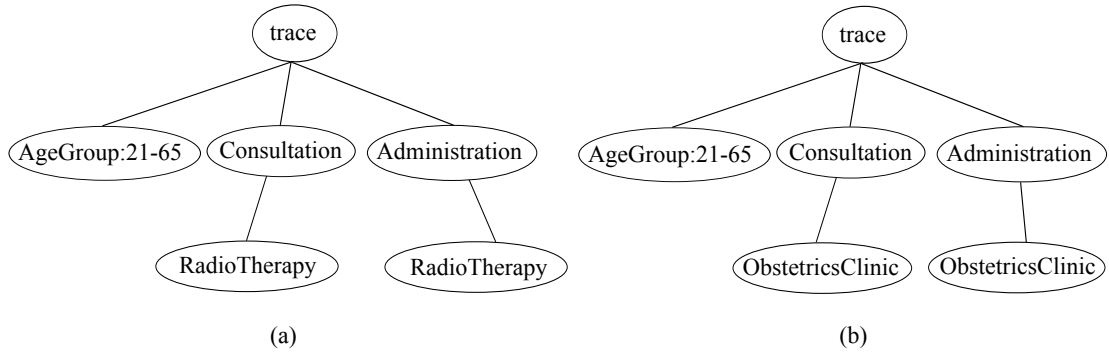


Figure 3.3: Two frequent closed subtrees of T_{HOS} with support=2.

3.1.2 Postion-constrained Subtrees

The order of events within a frequent path of execution is consistent among the matched instances in the database. However, the positional information of those individual events is not indicated in Fig. 3.3. The pattern in Fig. 3.3(b) tells us that the *Consultation* and *Administration* event occur frequently in at least two process instances. However, their specific orders of occurrence are unidentified, e.g. does the event occur at the beginning of the trace or at the end of that trace? Furthermore, it is not known whether these two events occur next to each other. If the position of the frequent events could be identified—for example, it can be learned from the database that *Consultation* frequently happens as the first event and *Administration* as the second event in the many processes—important observations can be drawn, for

instance, a hospital manager might be able conclude that current process executions comply to the hospital's recommended clinical pathways.

Traditional frequent pattern mining algorithms often face the problem of combinatorial explosion as "the number of frequent subtrees grows exponentially with the tree size" as stated in (Chi et al., 2004c). This phenomenon may also happen when mining with lower minimum support thresholds since many smaller size frequent subtrees can be found. In this situation, even when a machine is able to handle the complexity, the number of frequent subtrees are prohibitively large to be managed by users.

To alleviate the complexity associated with mining complex structures and to enable the direct application of a wider range of data analysis/mining techniques to tree-structured data, a position-constrained frequent subtree mining method was recently proposed in (Hadzic, 2012). The main idea of the method is that tree data can be converted to a flat representation using a *Database Structure Model (DSM)*. This representation can be converted to itemset format and a frequent itemset mining algorithms is used to obtain frequent itemsets representing subtrees. The experimental results from the study of (Hadzic et al., 2015) conclude that this method has "better efficiency for lower support thresholds and for higher structural complexity settings, and better scalability for larger datasets". One reason for that is by attaching positional information to each node, the complexity caused by nodes' label repetitions in each tree instance is eliminated. Even when repetitions in both nodes' labels and positions occur across tree instances, it is possible to remove the repetitions when the data is in the itemset format and the removed nodes can be reconstructed (using *DSM*) in the post-processing stage.

Another interesting implication of the *DSM* method is that positional information of nodes can be useful in process analysis as events are distinguished based on their exact occurrence within a trace of events. These characteristics and the ability to utilise a variety of data mining methods for different application aims, motivate us to use this approach as the basis of a method proposed in Section 3.4. It is unknown how this approach scales up against other traditional tree/sequence mining methods when applying to event data. Therefore, a time performance study is deemed necessary to show the feasibility of the approach in mining event logs.

3.2 The Structure-preserving Tree Mining Approach

In this section, *PCFSM*, an event data analysis approach that is based on the structure-preserving conversion of *XML*/tree data into a tabular/itemset format, initially introduced in (Hadzic, 2012), is proposed. From this representation, a variety of traditional data mining techniques can be directly applied to the data. The methods selected depend on specific business aims. As such, the process logs are analysed without the pre-requisite of a process model. This is in contrast to the majority of process analysis

techniques proposed in the literature, which focus on process model discovery, or use process models for further analysis. In general, the proposed approach contributes to the process mining field an integrated method for mining *XML*-based event logs.

The structure-preserving frequent subtree mining approach, described in Fig. 3.4, has three main steps: *DSM* extraction, conversion of the tree database to flat representation (tabular or itemset format), and application of data mining techniques. The detail of each step is discussed in Section 3.2.1, 3.2.2, and 3.2.3. The main concepts used in the approach are summarised in Section 3.2.4. Finally, *DSM* with a minimum support is described in Section 3.2.5.

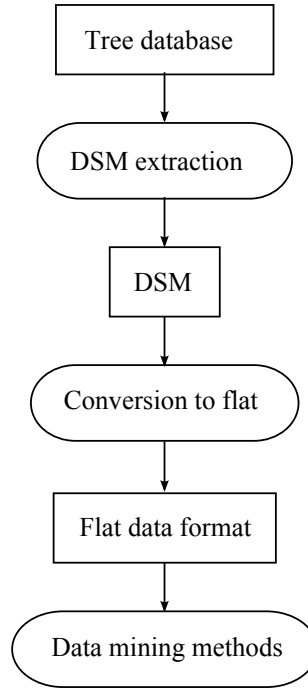


Figure 3.4: The structure-preserving tree mining approach.

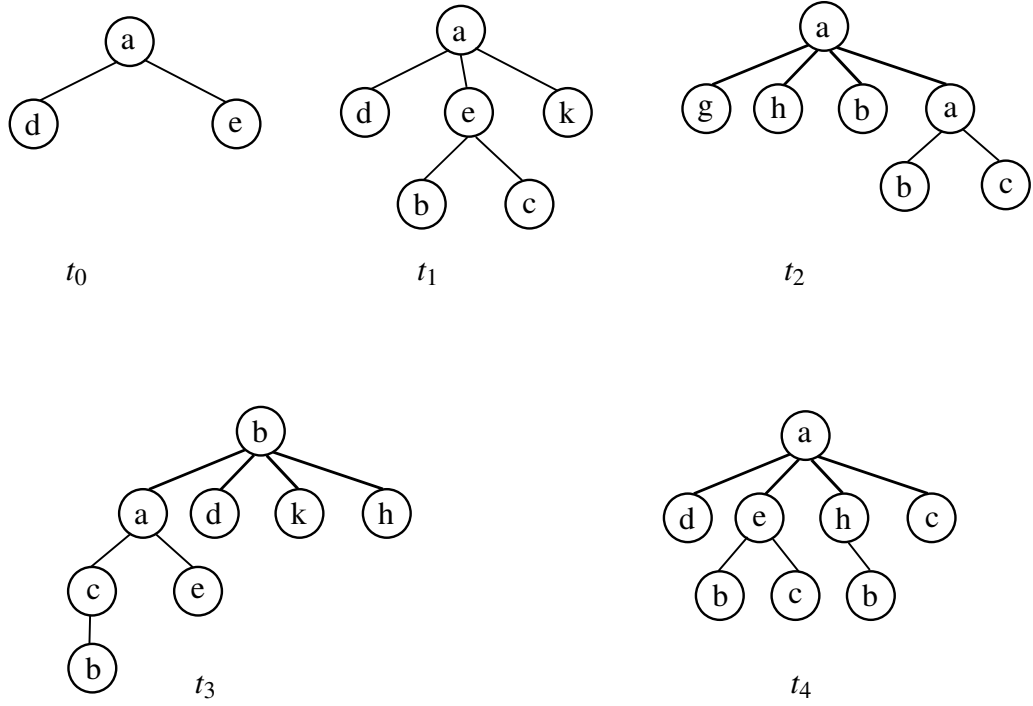
3.2.1 *DSM* extraction

In this section, the *DSM* structure and the *DSM* extraction algorithm are described. A synthetic tree database T_{EX} of rooted ordered labelled trees, shown in Fig. 3.5, is created for explanatory purpose.

The *DSM* Structure

DSM is a structure that captures the structural properties of all instances in a tree database, so that each node in a tree instance is structurally and uniquely matched to a node in the *DSM*.

To define *DSM*, some related concepts should be first introduced, such as *sibling order*, *top-left subtree*, and *candidate DSM*.


 Figure 3.5: Tree-structured database T_{EX} .

Definition 3.1 (Sibling order). Let x, y be two nodes of a tree, and y be the parent of node x . A sibling order of x , denoted by $\alpha(x)$, is the left-to-right order of x among other children of y . If x is a root node then $\alpha(x) = 1$.

Definition 3.2 (Top-left subtree). Given two trees: $S = (V_S, V_{S0}, E_S)$ and $T = (V_T, V_{T0}, E_T)$, where

- V_S and V_T are sets of vertices of S and T , respectively;
- V_{S0} and V_{T0} are root nodes of S and T , respectively;
- E_S and E_T are set of edges of S and T , respectively.

S is a top-left subtree of T , denoted by $S \preceq_{tl} T$, if and only if there exists an injective function $\zeta: V_S \rightarrow V_T$, which is called the top-left mapping function, such that

- $(x, y) \in E_S$ if and only if $(\zeta(x), \zeta(y)) \in E_T$;
- $x \in V_S$, $\alpha(x) = k$ if and only if $\alpha(\zeta(x)) = k$;
- $\zeta(V_{S0}) = V_{T0}$.

An example of top-left subtree is shown in Fig. 3.6 where S is a top-left subtree of T and T is a top-left super-tree of S . The characteristics of tree S and tree T are described as follows.

- $V_{S0} = S_1$, $V_{T0} = T_1$;

- $V_S = \{S_1, S_2, S_3, S_4\}$, $V_T = \{T_1, T_2, T_3, T_4\}$;
- $E_S = \{(S_1, S_2), (S_1, S_3), (S_3, S_4)\}$;
- $E_T = \{(T_1, T_2), (T_2, T_3), (T_1, T_4), (T_4, T_5), (T_4, T_6), (T_1, T_7)\}$;
- $\zeta(S_1) = T_1$, $\zeta(S_2) = T_2$, $\zeta(S_3) = T_4$, $\zeta(S_4) = T_5$;
- $\alpha(S_1) = 1$, $\alpha(S_2) = 1$, $\alpha(S_3) = 2$, $\alpha(S_4) = 1$;
- $\alpha(T_1) = 1$, $\alpha(T_2) = 1$, $\alpha(T_4) = 2$, $\alpha(T_5) = 1$.

Based on the description of trees S and T , it can be concluded that S is a top-left subtree of T .

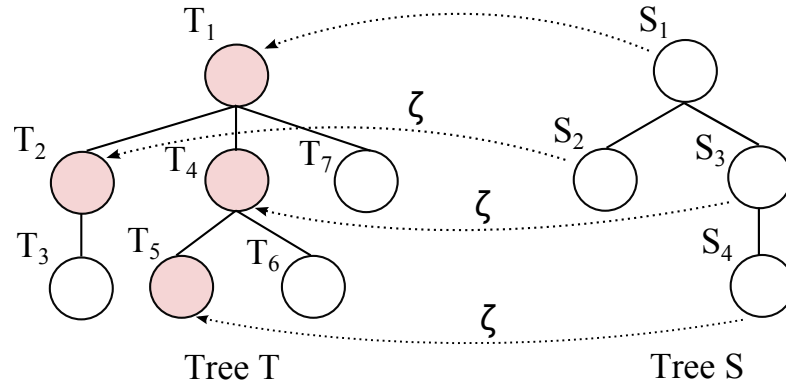


Figure 3.6: An example of top-left subtree relation.

The fundamental properties of the top-left subtree relation are given as follows.

Property 3.3 (Transitivity). *The top-left subtree relation is transitive: If $S \preceq_{tl} T$ and $T \preceq_{tl} W$ then $S \preceq_{tl} W$.*

Proof. The transitivity property can be proved by showing that from $S \preceq_{tl} T$ and $T \preceq_{tl} W$. Thus, $S \preceq_{tl} W$.

Let $S = (V_S, V_{S0}, E_S)$, $T = (V_T, V_{T0}, E_T)$, and $W = (V_W, V_{W0}, E_W)$.

Since $S \preceq_{tl} T$, there exists an injective top-left mapping function $\zeta_1: V_S \rightarrow V_T$ such that

- $(x, y) \in E_S$ if and only if $(\zeta_1(x), \zeta_1(y)) \in E_T$; (1)
- $x \in V_S$, $\alpha(x) = k$ if and only if $\alpha(\zeta_1(x)) = k$; (2)
- $\zeta_1(V_{S0}) = V_{T0}$. (3)

Since $T \preceq_{tl} W$, there exists an injective top-left mapping function $\zeta_2: V_T \rightarrow V_W$ such that

- $(x, y) \in E_T$ if and only if $(\zeta_2(x), \zeta_2(y)) \in E_W$; (4)

- $x \in V_T, \alpha(x) = k$ if and only if $\alpha(\zeta_2(x) = k)$; (5)
- $\zeta_2(V_{T0}) = V_{W0}$. (6)

Let $\zeta = \zeta_2 \circ \zeta_1$ be the composite top-left mapping function.

Let $u = \zeta_1(x)$ and $v = \zeta_1(y)$, (1) becomes $(x, y) \in E_S$ if and only if $(u, v) \in E_T$

$\Rightarrow (x, y) \in E_S$ if and only if $(\zeta_2(u), \zeta_2(v)) \in E_W$ (based on (4))

$\Rightarrow (x, y) \in E_S$ if and only if $(\zeta_2(\zeta_1(x)), \zeta_2(\zeta_1(y))) \in E_W$.

$\Rightarrow (x, y) \in E_S$ if and only if $(\zeta(x), \zeta(y)) \in E_W$ (7)

In the same manner, the following predicates can be proved.

- $x \in V_S, \alpha(x) = k$ if and only if $\alpha(\zeta(x) = k)$ (8)
- $\zeta(V_{S0}) = V_{W0}$ (9)

From (7), (8), and (9), it can be concluded that $S \preceq_{tl} W$ ■

Property 3.4 (Reflexitivity). *The top-left subtree relation is reflexive: $\forall S : S \preceq_{tl} S$.*

Proof. The reflexivity property can be proved by showing that $\forall S : S \preceq_{tl} S$.

Let $S = (V_S, V_{S0}, E_S)$. There exists a top-left mapping function $\zeta : V_S \rightarrow V_S$, such that

- $\zeta(x) = x$;
- $(x, y) \in E_S$ if and only if $(\zeta(x), \zeta(y)) \in E_S$; (1)
- $x \in V_S, \alpha(x) = k$ if and only if $\alpha(\zeta(x) = k)$; (2)
- $\zeta(V_{S0}) = V_{S0}$. (3)

From (1), (2), and (3), it can be concluded that $S \preceq_{tl} S$ ■

The top-left mirror concept defined below will be used for the assignment of positions to tree nodes.

Definition 3.5 (Top-left mirror). *Given $S \preceq_{tl} T$ and $S = (V_S, V_{S0}, E_S)$, a top-left mirror of S on T , denoted by $\mu_T(S)$, are a set of nodes and edges which are the mappings of all nodes and edges of S on T using the top-left mapping function $\zeta : V_S \rightarrow V_T$. We have $\mu_T(S) = (V, V_0, E)$, where*

- $V = \{\zeta(x) | \forall x \in V_S\}$;
- $V_0 = \zeta(V_{S0})$;
- $E = \{(\zeta(u), \zeta(v)) | \forall (u, v) \in E_S\}$.

Given that S is a top-left subtree of T as shown in Fig. 3.6, the top-left mirror of S on T is $\mu_T(S) = \{V, V_0, E\}$ where

- $V = \{\zeta(S_1), \zeta(S_2), \zeta(S_3), \zeta(S_4)\} = \{T_1, T_2, T_4, T_5\};$
- $V_0 = \zeta(V_{S0}) = \zeta(S_1) = T_1;$
- $E = \{(\zeta(S_1), \zeta(S_2)), (\zeta(S_1), \zeta(S_3)), (\zeta(S_3), \zeta(S_4))\} = \{(T_1, T_2), (T_1, T_4), (T_4, T_5)\}.$

Property 3.6 (Top-left mirror). *Let $\mu_T(S) = \{V, V_0, E\}$ be a top-left mirror of $S = (V_S, V_{S0}, E_S)$ on T , $\mu_T(S)$ is a connected tree.*

Proof. Since all nodes in V_S are connected by edges in E_S , all nodes in V are connected by edges in E ■

In Fig. 3.6, the top-left mirror of S on T is the subtree of T whose nodes are shaded. The definition of candidate *DSM* and its properties are presented as follows.

Definition 3.7 (Candidate *DSM*). *Let TDB be a tree database. A set of candidate *DSM* trees of TDB, denoted by Δ , is defined as:*

$$\Delta = \{d | \forall t \in TDB : t \preceq_{tl} d\}.$$

Property 3.8 (Candidate *DSM*). *Let TDB be a tree database, and Δ be a set of candidate *DSM* trees of TDB. If D is a top-left super-tree of a candidate *DSM* of TDB then D is also a candidate *DSM* of TDB. This property is formally described as follows.*

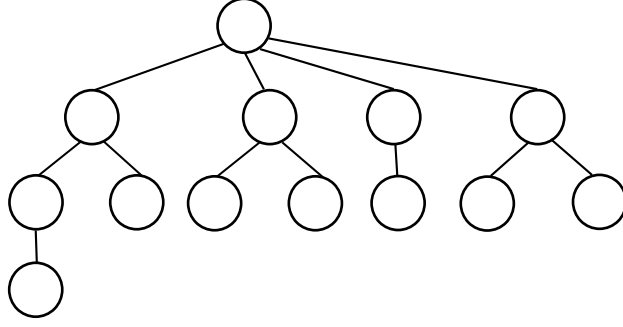
$$d \in \Delta \text{ and } d \preceq_{tl} D \Rightarrow D \in \Delta.$$

From a candidate *DSM*, the *DSM* can be extracted by removing nodes that are not mapped by any node from the tree instances using the top-left mapping function. Another method to obtain the *DSM* from a candidate *DSM* is to take the union of the top-left mirror of all database instances on the candidate *DSM*. The *DSM* is defined as follows.

Definition 3.9 (*DSM*). *Let TDB be a tree database, and Δ be a set of candidate *DSM* trees of TDB. Tree $\delta = (V_\delta, V_{\delta 0}, E_\delta)$ is the *DSM* of TDB if and only if*

- $\delta \in \Delta$ (δ is a candidate *DSM* of TDB);
- $\forall x \in V_\delta, \exists t = (V_t, V_{t0}, E_t) \in TDB, \exists u \in V_t : \zeta_t(u) = x$. Note that ζ_t is the top-left mapping function of t on δ .

The *DSM* of the example database T_{EX} is shown in Fig. 3.7 and the top-left mirrors of the tree instances on the *DSM* of T_{EX} are shown in shaded regions in Fig. 3.8.


 Figure 3.7: The DSM of T_{EX} .

The DSM structure is intended to be used as a map in which tree nodes can be located precisely. For this purpose, each node is identified by its position in the pre-order traversal of the DSM . Each edge is named by the order of backtracking from the corresponding child node to its parent node during the traversal. Note that the backtracking data are not always shown in many DSM diagrams for the sake of brevity. In Fig. 3.9, each node and edge of the DSM of T_{EX} are named by their position and backtracking order, respectively.

Property 3.10 (DSM). *Let TDB be a tree database having only one instance d and δ is the DSM of TDB . Then, $\delta = d$.*

Proof. The reflexivity property of the top-left subtree relation leads to $d \preceq_{tl} d$ (1). From $TDB = \{d\}$ and (1), it can be concluded that $d \in \Delta$ (based on the definition of candidate DSM) (2). Since the top-left mirror of d is itself, it can be inferred that $\forall x \in V_d, \exists d \in TDB : x \in V_{\mu_d(d)}$ (3). From (2) and (3), $\delta = d$. ■

Property 3.11 (DSM). *Let TDB be a tree database, and $\delta = (V_\delta, V_{\delta 0}, E_\delta)$ be the DSM of TDB . Then, $\forall (x, y) \in E_\delta, \exists t = (V_t, V_{t0}, E_t) \in TDB, \exists (u, v) \in E_t : (\zeta_t(u), \zeta_t(v)) = (x, y)$. In other words, for every edge in the DSM , there exists a top-left mapping from the edge of one or more trees. Note that $\zeta_t : V_t \rightarrow V_\delta$ is the top-left mapping function of t on δ .*

Proof. It is safe to assume that there is node y which is the parent of x . By the definition of DSM , $\forall x \in V_\delta, \exists t = (V_t, V_{t0}, E_t) \in TDB, \exists u \in V_t : \zeta_t(u) = x$.

Let node v be the parent of node u . The top-left mapping function ζ_t maps v to node $k \in V_\delta$. Since $(u, v) \in E_t$, we have $(x, k) \in E_\delta$. If $k \neq y$, then k must be a child of x (because y is already the parent of x and x cannot have two parents). In Fig. 3.10, it is assumed that k is a child of x . If v is the root of tree t , then t cannot be a top-left subtree of δ because $\alpha(v) = 0$ while $\alpha(k) > 0$. For any node that is a parent of v , its top-left mapping to δ is a child of $\zeta_t(v)$. From this, it can be seen that the mapping of the root node of tree t is a leaf node of δ , which cannot be the case because it violates the definition of top-left subtree. Therefore $k = y$. In other words, for all $(x, y) \in E_\delta$ there exists $(u, v) \in E_t$, such that $(\zeta_t(u), \zeta_t(v)) = (x, y)$. ■

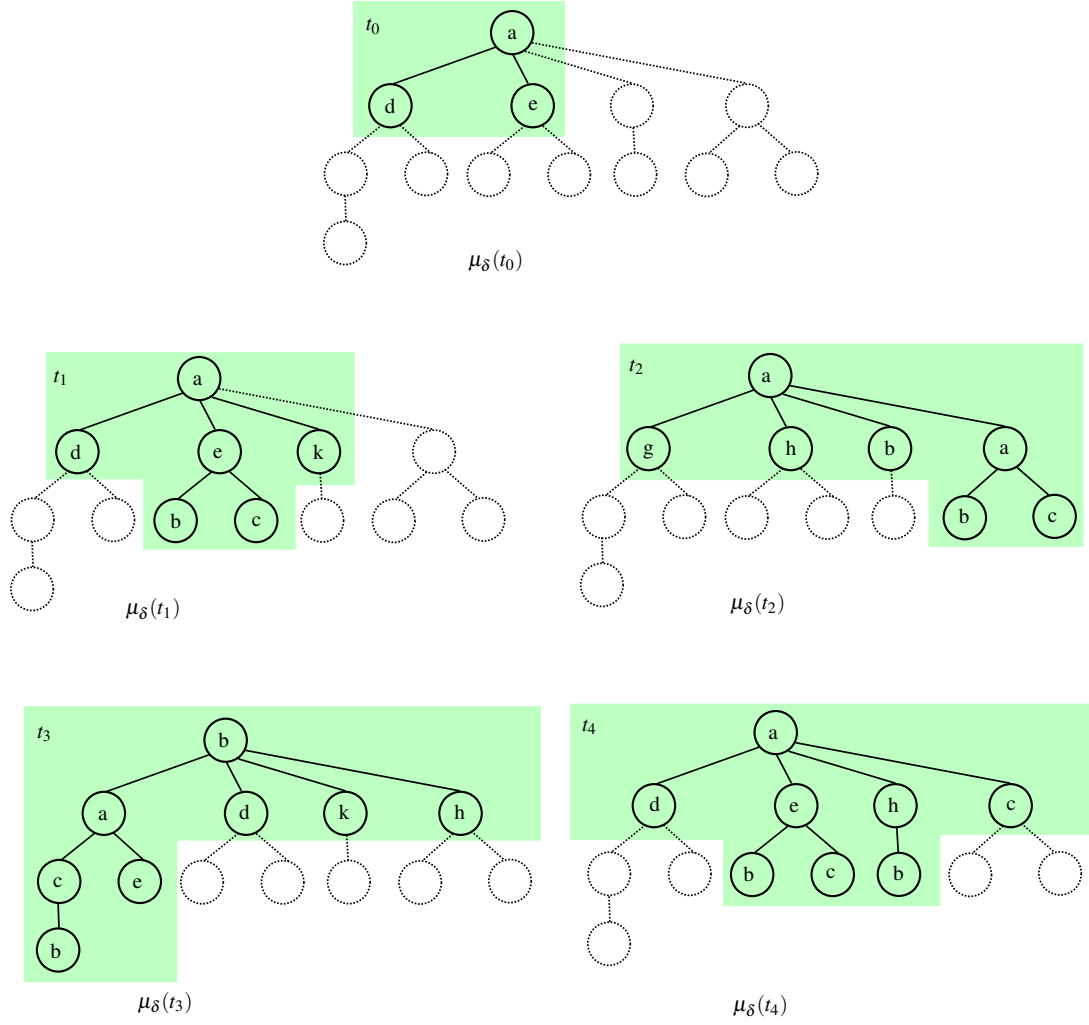


Figure 3.8: The top-left mirror of tree instances on the DSM of T_{EX} .

From the definition of DSM , it can be seen that all DSM are isomorphic (Zaki, 2005a). This property indicates that all trees that are DSM of a tree database are isomorphic.

Property 3.12 (DSM). *Let TDB be a tree database. If both δ_1 and δ_2 are the DSM of TDB then δ_1 and δ_2 are isomorphic.*

Proof. Let $\delta_1 = (V_1, V_{10}, E_1)$ and $\delta_2 = (V_2, V_{20}, E_2)$. By Property 3.11, $\forall (x, y) \in E_1$, $\exists t = (V_t, V_{t0}, E_t) \in TDB, \exists (u, v) \in E_t : (\zeta_{t1}(u), \zeta_{t1}(v)) = (x, y)$. Note that $\zeta_{t1} : V_t \rightarrow V_1$ is the top-left mapping function of t on δ_1 . In short, $\exists t \in TDB: (x, y) \in E_1$ if and only if $(\zeta_{t1}^{-1}(x), \zeta_{t1}^{-1}(y)) \in E_t$. (1)

By the definition of DSM , t is a top-left subtree of δ_2 . Given ζ_{t2} as a top-left mapping function from t to δ_2 , $(u, v) \in E_t$ if and only if $(\zeta_{t2}(u), \zeta_{t2}(v)) \in E_2$. (2)

Based on (1) and (2), we have $(x, y) \in E_1$ if and only if $(\zeta_{t2}(\zeta_{t1}^{-1}(x)), \zeta_{t2}(\zeta_{t1}^{-1}(y))) \in E_2$. By the definition of isomorphic subtree, $\delta_1 \preceq \delta_2$.

The same method can be applied to prove that $\delta_2 \preceq \delta_1$. By definition of *isomorphic* in (Zaki, 2005a), δ_1 and δ_2 are isomorphic. ■

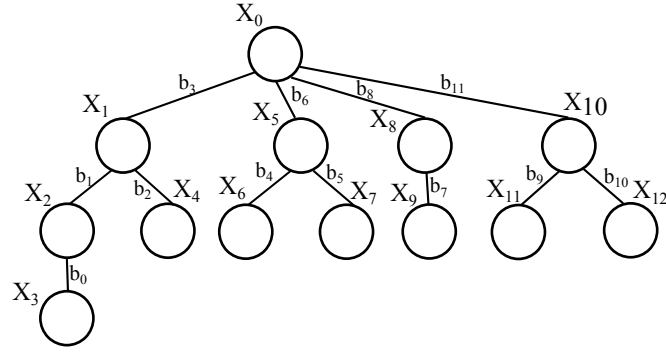


Figure 3.9: Each node and edge of the *DSM* of T_{EX} is named by their position and backtracking order.

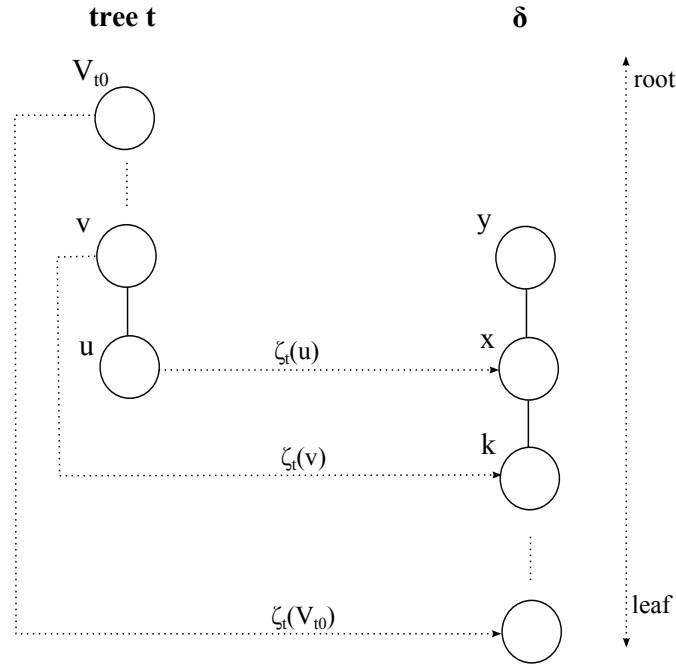


Figure 3.10: An illustration for the proof of 3.11.

DSM Extraction Algorithm

The pseudo-code of the algorithms to extract the *DSM* (δ) from a tree database *TDB* are shown in Algorithm 1 and 2. Suppose that each there are N event nodes in each trace, and there are M attributes in each event. The complexity of Algorithm 1 and 2 is $O(TDB * M * N)$.

Another method to extract the *DSM* (δ) from *TDB* that is already in pre-order string encoding is presented in Algorithm 3. The description of the pseudo-code is given below (Hadzic et al., 2015).

The *DSM* is a tree where the set of nodes V_δ is such that the pre-order position of each $v_i \in V_\delta$ is reflected through its unique identifier i ; moreover, v_i has a set $M(v_i)$ of mappings where each $m \in M(v_i)$ is a pair $(L_{T_j}(v_i), t)$ that identifies a node (denoted as $T_{j \cdot v_i}$) with label $L_{T_j}(v_i)$ that belongs to a

Algorithm 1 Discover *Database Structure Model* by joining trees.

Input: a tree database TDB

Output: the DSM tree (δ)

```

1:  $jt = \emptyset$   $\triangleright$   $jt$  is a tree
2: for  $i = 1$  to  $|TDB|$  do
3:    $jt = topleftmerge(jt, v_i)$   $\triangleright v_i$  is  $i^{th}$  tree in  $TDB$ 
4:  $PreorderTraversal(jt)$   $\triangleright$   $jt$ 's nodes are labelled by their traversal orders return
    $jt$ 

```

Algorithm 2 $topleftmerge(v_1, v_2)$.

Input: trees v_1, v_2

Output: a merged tree from v_1, v_2

```

1:  $s_1 = |v_1.children|$ 
2:  $s_2 = |v_2.children|$ 
3: if  $s_1 = 0$  and  $s_2 = 0$  then return new tree node
4: else if  $s_1 = 0$  then return  $v_2$ 
5: else if  $s_2 = 0$  then return  $v_1$ 
6: else
7:    $t =$  new tree node
8:    $minsize = \min(s_1, s_2)$ 
9:   for  $i = 1$  to  $minsize$  do
10:     $t.addchild(topleftmerge(v_1.child(i), v_2.child(i)))$ 
11:   if  $s_1 < s_2$  then
12:     for  $i = minsize$  to  $s_2$  do
13:        $t.addchild(v_2.child(i))$ 
14:   if  $s_2 < s_1$  then
15:     for  $i = minsize$  to  $s_1$  do
16:        $t.addchild(v_1.child(i))$ 
return  $t$ 

```

tree $T_j \in TDB$, with $j = t$. Let the set of child nodes of a node v_i be denoted as $children(v_i)$, and similarly the child nodes of the corresponding node in T_j as $T_j.children(v_i)$. Note that for each node in DSM , the size of its mapping set indicates the number of times that a tree instance has matched that node in DSM .

Applying the DSM extraction algorithm to the pre-order string encoding of the tree database T_{EX} (shown in Table 3.1), the pre-order string encoding of $\delta_{T_{EX}}$ can be obtained: ‘ $X_0 X_1 X_2 X_3 b_0 b_1 X_4 b_2 b_3 X_5 X_6 b_4 X_7 b_5 b_6 X_8 X_9 b_7 b_8 X_{10} X_{11} b_9 X_{12} b_{10} b_{11}$ ’. Note that $b_i, X_i, i=\{0, 1, \dots, n-2\}$ (n is the size of the DSM), denotes the backtrack symbol ‘ -1 ’ and the pre-order positions of the nodes in DSM , respectively. Based on this string encoding, the DSM shown in Fig. 3.7 or Fig. 3.9 can be easily obtained.

Algorithm 3 *Database Structure Model (DSM) extraction* (reproduced from (Hadzic et al., 2015)).

Input: a tree database TDB

Output: the DSM tree (δ)

```

1:  $V_\delta := \emptyset, M_\delta := \emptyset$  ▷ Node set and node mapping set of DSM
2:  $E_\delta := \emptyset$  ▷ Edge set of DSM
3:  $Q := \emptyset, C := \emptyset$ 
4:  $i := 1, Q.append(v_i)$  ▷ Node queue initially containing the root node
5: while  $Q \neq \emptyset$  do
6:    $v_i = Q.remove(), V_\delta := \cup v_i$ 
7:    $M(v_i) := \bigcup_{T_j \in TDB} (L_{T_j}(v_i), j)$ 
8:    $M_\delta := M_\delta \cup M(v_i)$ 
9:    $E_\delta := E_\delta \cup \{C.parent(v_i), v_i\}$ 
10:  if  $(\exists T_j \in TDB \text{ s.t. } T_j.children(v_i) \neq \emptyset)$  then
11:     $T_j = \operatorname{argmin}_{T_h} \{T_h \in TDB \mid |T_h.children(v_i)| \geq |T'.children(v_i)|, T_h \neq T'\}$ 
12:     $children(v_i) := \emptyset$ 
13:    for  $c = 1$  to  $|T_j.children(v_i)|$  do
14:       $children(v_i) := children(v_i) \cup \{v_{i_c}\}$ 
15:       $C := C \cup (i, children(v_i))$ 
16:       $i := i + 1, Q.append(v_i)$ 
17:  else
18:    while  $(!C.parent(v_i).hasNextChild())$  do
19:       $v_i := C.parent(v_i)$ 
20:       $Q.append(C.parent(v_i).nextChild())$ 
21:   $i := i + 1$ 
return  $\delta = (V_\delta, E_\delta, M_\delta)$ 

```

3.2.2 Conversion to Flat Data Format

In this section, two alternatives of flat data format are used to represent a tree, i.e. the tabular and the itemset format. These representations can be converted to one another easily without loss of any information. In the following subsections, firstly a conversion from trees to itemset format is introduced, secondly a conversion from itemset format to tabular format is given, and finally an algorithm to directly generate flat representation from tree instances is described.

Converting Trees to Itemset Format

For each tree in the database, its top-left mirror on the DSM ($\mu_\delta(t)$) is identified. The labels of the top-left mirror are then matched and merged with the position information of the DSM . It is known that $\mu_\delta(t)$ is a connected subtree and is part of the DSM . Since all nodes of t should be identified by their location in the DSM , the top-left mirror of t on the DSM are labelled according to the labels of their corresponding nodes in t . The resulting top-left mirror structure is called *Position-constrained Top-left Mirror (PCT)*.

Table 3.1: pre-order string encoding of T_{EX} .

T_{EX}	pre-order string encoding
t_0	a d -1 e
t_1	a d -1 e b -1 c -1 -1 k -1
t_2	a g -1 h -1 b -1 a b -1 c -1 -1
t_3	b a c b -1 -1 e -1 -1 d -1 k -1 h -1
t_4	a d -1 e b -1 c -1 -1 h b -1 -1 c -1

Definition 3.13 (*Position-constrained Top-left Mirror (PCT)*). Let $t = (V_t, V_{t0}, E_t, L_t)$ be a tree instance of TDB , $\delta = (V_\delta, V_{\delta0}, E_\delta, L_\delta)$ be the DSM of TDB , and ζ be the top-left mapping function that maps V_t to V_δ . A position-constrained top-left mirror (PCT) of t on δ , denoted as $PCT_\delta(t)$ (or $PCT(t)$), is a tree $(V_{PCT}, V_{PCT0}, E_{PCT}, L_{PCT})$ for which the following conditions hold.

1. $V_{PCT} = \{v | v = \zeta(x) \text{ and } x \in V_t\}$
2. $E_{PCT} = \{(u, v) | u = \zeta(x), v = \zeta(y) \text{ and } (x, y) \in E_t\}$
3. $\forall u \in V_{PCT} : \alpha(u) = k$ if and only if $\exists x : u = \zeta(x) \text{ and } \alpha(x) = k$
4. $V_{PCT0} = \zeta(V_{t0})$.
5. $\forall u \in V_{PCT}, \exists x \in V_t : u = \zeta(x) \text{ and } L_{PCT}(u) = L_t(x)$.

Fig. 3.11 shows the PCT s of all tree instances of T_{EX} on the DSM. Note that the PCT s are shaded subtrees.

Tree instances in the form of PCT are next converted to an itemset representation. From each PCT , a set of *position-constrained nodes* are extracted, each of which comprises of a label and a position. This set of position-constrained nodes is called $FPCT$. For example, the *Flat Position-constrained Top-left Mirror (FPCT)* of tree t_1 on the DSM is ‘ $aX_0, dX_1, eX_5, bX_6, cX_7, kX_8$ ’.

Definition 3.14 ($FPCT$). Let t be a tree in TDB , δ be the DSM of TDB , and $PCT_\delta(t) = (V_{PCT}, V_{PCT0}, E_{PCT}, L_{PCT})$. A $FPCT$ of tree t on the DSM of TDB , denoted as $FPCT_\delta(t)$ (or $FPCT(t)$), is defined as:

$FPCT(t) = \{p | p = L_{PCT}(x) + x, \forall x \in V_{PCT}\}$. Note that $+$ is the concatenation operator.

Conversion to Tabular Format

To utilise traditional data mining methods, e.g. decision trees, clustering, the $FPCT$ s of the tree database should be converted to tabular format. Let $F(C, R)$ be the table

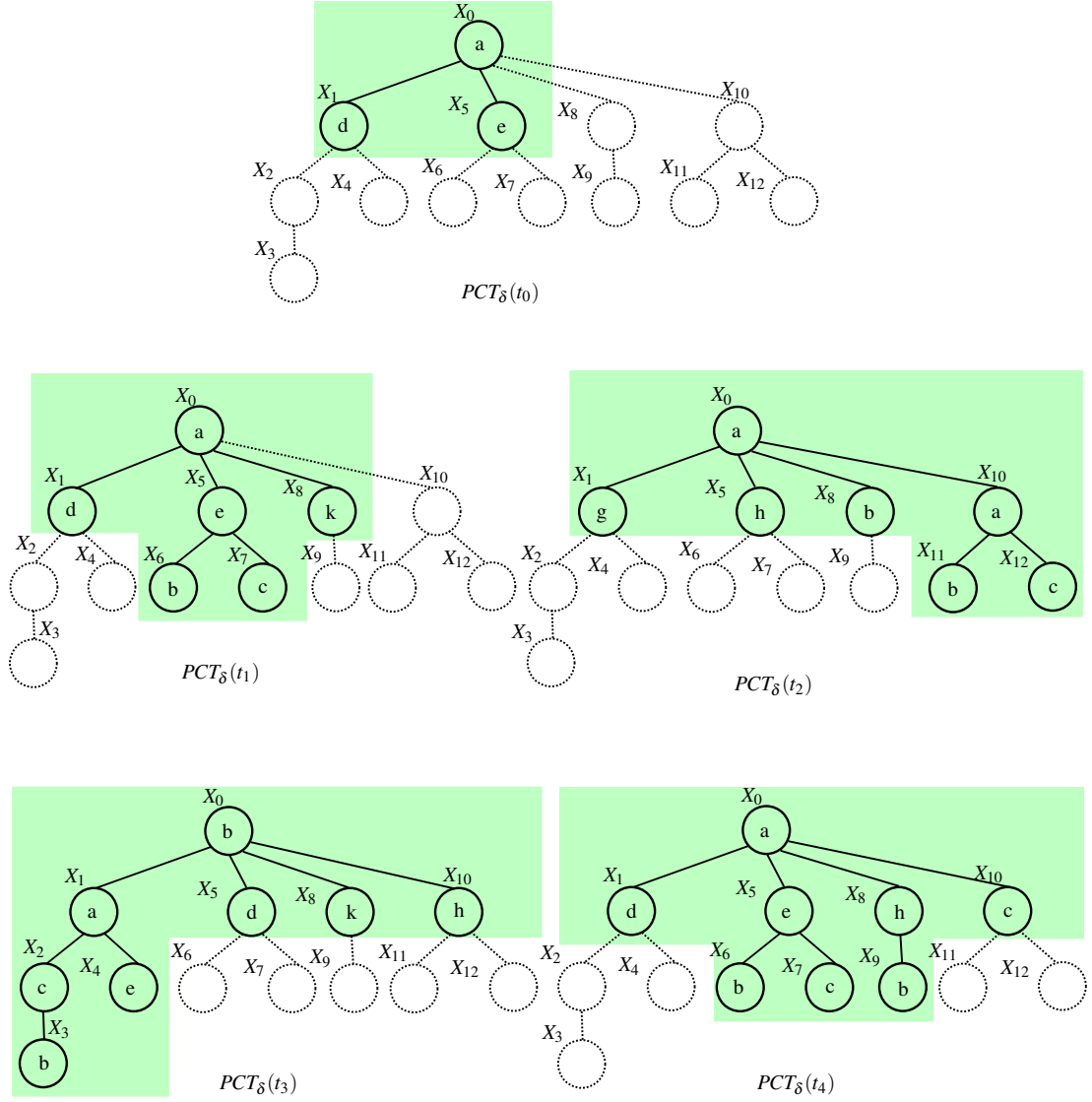


Figure 3.11: The position-constrained top-left mirrors (shaded areas) of trees on the *DSM*.

of C columns and R rows, where $C = \{c_i\}$, $i = 0 \rightarrow |\phi(\delta_{DB}) - 1|$, and $R = \{r_j\}$, $j = 0 \rightarrow |DB|$. The column names of table F are: $F(c_i, r_0) = \phi(\delta_{DB})_i$, where $i = 0 \rightarrow |\phi(\delta_{DB}) - 1|$. For example, the column names of the tabular form of the tree database T_{EX} is ‘ $X_0 X_1 X_2 X_3 b_0 b_1 X_4 b_2 b_3 X_5 X_6 b_4 X_7 b_5 b_6 X_8 X_9 b_7 b_8 X_{10} X_{11} b_9 X_{12} b_{10} b_{11}$ ’. From each position-constrained node of a *FPCT*, e.g. aX_0 , the node value (a) is extracted and put under the corresponding columns (X_0). The tabular format of t_1 is shown at the second row of Table 3.2. Note that the backtrack symbol ‘-1’ is represented by a value of ‘1’, which is put under the corresponding backtrack column. At columns where there are no corresponding position-constrained nodes or backtrack values, a value of ‘0’ is assigned. The tabular format ($F(C, R)$) of the tree database is called *Flat Data Representation (FDT)*.

An Algorithm to Convert Tree Database to Tabular Format

Previous subsections describe the formal method of converting tree representation to tabular format. The pseudo-code of the algorithm to perform the conversion is shown in Algorithm 4, where each column name is one element in the pre-order string encoding of the DSM , and the backtrack symbols (-1) are replaced by b_i . The main idea of the algorithm is to synchronously traverse the pre-order string encoding of the DSM and the tree instance. The two variables $inputNodeLevel$ and $DSMNodeLevel$ are used to keep track of the current position of the traversals.

Algorithm 4 *The conversion of tree-structured data to FDT format (Hadzic, 2012).*

Input: TDB , the DSM tree (δ)

Output: F

```

1:  $F(c_i, r_0) = \phi(\delta)_k$  ▷ set up the attribute name row in  $F$ 
2:  $inputNodeLevel := 0$  ▷ current level of  $\phi(t_i)_k$ 
3:  $DSMNodeLevel := 0$  ▷ current level of  $\phi(\delta)_k$ 
4: for  $i = 0$  to  $n - 1$  do ▷ populate  $F$ 
5:   for each  $\phi(t_i)_k \in \phi(t_i)$  do
6:     for  $p = 0$  to  $(|\phi(\delta)| - 1)$  do
7:       if  $\phi(t_i)_k = -1$  then
8:          $inputNodeLevel --$ 
9:       else
10:         $inputNodeLevel ++$ 
11:      if  $\phi(\delta)_p = b_i$  then
12:         $DSMNodeLevel --$ 
13:      else
14:         $DSMNodeLevel ++$ 
15:      if  $inputNodeLevel = DSMNodeLevel$  then
16:        if  $\phi(t_i)_k = -1$  then
17:           $F(c_p, r_{i+1}) := 1$ 
18:        else
19:           $F(c_p, r_{i+1}) := \phi(t_i)_k$ 
20:      else ▷ level mismatch, traverse  $\phi(\delta)$  until match
21:        while  $inputNodeLevel \neq DSMNodeLevel$  do
22:           $F(c_p, r_{(i+1)}) := 0$ 
23:           $p ++$ 
24:          if  $\phi(\delta)_p = b_i$  then
25:             $DSMNodeLevel --$ 
26:          else
27:             $DSMNodeLevel ++$ 
28:          if  $\phi(t_i)_k = -1$  then
29:             $F(c_p, r_{(i+1)}) := 1$ 
30:          else
31:             $F(c_p, r_{i+1}) := \phi(t_i)_k$ 
return  $F$ 
    
```

Table 3.2: The *FDT* format of T_{Ex} .

t	X_0	X_1	X_2	X_3	b_0	b_1	X_4	b_2	b_3	X_5	X_6	b_4	X_7	b_5	b_6	X_8	X_9	b_7	b_8	X_{10}	X_{11}	b_9	X_{12}	b_{10}	b_{11}	
t_0	a	d	0	0	0	0	0	0	1	e	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
t_1	a	d	0	0	0	0	0	0	1	e	b	1	c	1	1	k	0	0	0	1	0	0	0	0	0	
t_2	a	g	0	0	0	0	0	0	1	h	0	0	0	0	1	b	0	0	0	1	a	b	1	c	1	1
t_3	b	a	c	b	1	1	e	1	1	d	0	0	0	0	1	k	0	0	0	1	h	0	0	0	0	1
t_4	a	d	0	0	0	0	0	0	1	e	b	1	c	1	1	h	b	1	1	1	c	0	0	0	0	1

3.2.3 Data Mining Methods

From the tabular format (*FDT*) of a tree database, a wide range of data mining methods such as itemset mining, association rule mining, classification, clustering, outlier detection can be utilised. In this chapter, itemset mining is given as an example.

Itemset Mining

As discussed in Section 3.2.2, the *PCTs* of tree instances correspond to transactions in market basket data (see Section 2.3.3) where each position-constrained node corresponds to an item in the transaction. One can obtain the itemset format from the tabular representation of a tree database by attaching node values with their corresponding position (the column name) and removing all non-existing nodes (nodes having value of '0') and backtrack values. For example, the itemset format corresponding to the tabular representation shown in Table 3.2 is presented in Table 3.3.

Table 3.3: The itemset format of T_{EX} .

T_{EX}	Itemsets
t_0	aX_0, dX_1, eX_5
t_1	$aX_0, dX_1, eX_5, bX_6, cX_7, kX_8$
t_2	$aX_0, gX_1, hX_5, bX_8, aX_{10}, bX_{11}, cX_{12}$
t_3	$bX_0, aX_1, cX_2, bX_3, eX_4, dX_5, kX_8, hX_{10}$
t_4	$aX_0, dX_1, eX_5, bX_6, cX_7, hX_8, bX_9, cX_{10}$

If a frequent itemset mining algorithm with a minimum support of 3 is applied to the database shown in Table 3.3, one of the frequent itemsets obtained is aX_0, eX_5 , which occurs in t_0, t_1 , and t_4 . The obtained frequent itemsets can be converted back into tree representation as described in the following subsection.

Conversion to Tree

Since *FPCT* is a set of position-constrained nodes, a frequent itemset mining algorithm applied to a set of *FPCTs* would result in itemsets which are also a set of position-constrained nodes, called *pFPCTs*. For example, the frequent itemset aX_0, eX_5 is called a *pFPCT*. It is possible to convert (or reconstruct) a *pFPCT* to a tree using *DSM* as a reference model.

Definition 3.15 (Reconstructed *pFPCT*). Let TDB be a tree database and $\delta = (V_\delta, V_{\delta 0}, E_\delta)$ be its *DSM*. Let $pFPCT = \{p | p \text{ is a position-constrained node}\}$ be a set of position-constrained nodes, such that (1) $\forall p_i, p_j \in pFPCT, i \neq j: \beta(p_i) \neq \beta(p_j)$, and (2) $\beta(p) \in [0..(|V_\delta| - 1)]$ where $\beta(p)$ is a function that return the position of a position-constrained node e.g., $\beta(aX_0) = 0$. Let $+$ is the concatenation operator.

The reconstructed tree of $pFPCT$ based on δ is a connected tree that is denoted by $\psi(pFPCT) = (V_\psi, V_{\psi 0}, E_\psi, L_\psi)$ and is defined as follows.

1. $V_\psi = \{v | v \in V_\delta, v = X + \beta(p), \forall p \in pFPCT\}$;
2. $E_\psi = \{(x, y) | \forall x, \forall y \in V_\psi, \nexists z \in V_\delta : x \leq_c y, c > 0, x \leq_d z \text{ and } d < c\}$;
3. $V_{\psi 0} = v \in V_\psi$ such that $\forall u \in V_\psi : u \neq v, v \leq_m u, m > 0$;
4. $L_\psi(v) = k$ if and only if $k + v \in pFPCT$.

Note that if condition 3 does not hold: $\nexists v \in V_\psi$ such that $\forall u \in V_\psi : u \neq v, v \leq_m u, m > 0$, $\psi(pFPCT)$ is a disconnected tree—there is at least one subtree that has no path to other nodes of the tree.

As an example, suppose that $pFPCT = \{aX_0, bX_6, cX_7\}$, the reconstructed tree from $pFPCT$ on the DSM of T_{EX} is shown Fig. 3.12. In another example, if $pFPCT = \{bX_2, cX_5, aX_6, dX_7, eX_{10}\}$, its reconstructed tree is *disconnected*, as shown in Fig. 3.13.

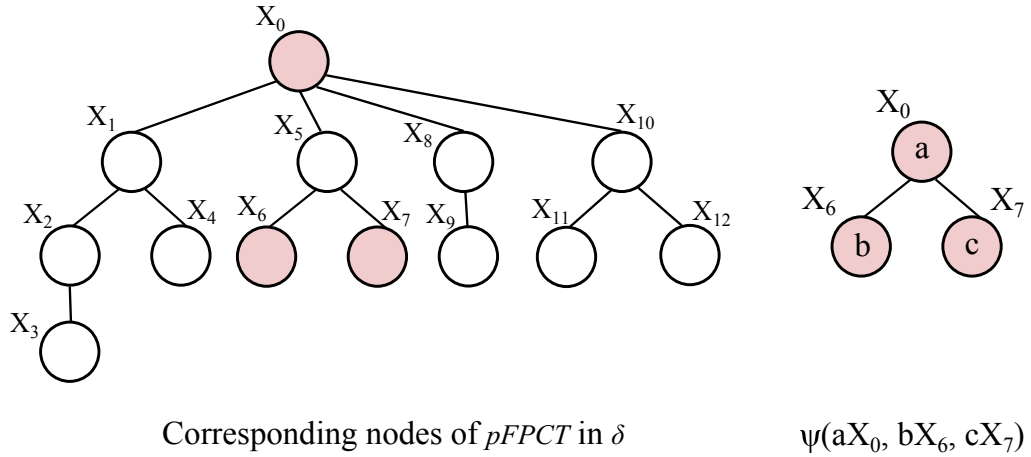


Figure 3.12: The reconstruction of $pFPCT = \{aX_0, bX_6, cX_7\}$.

The reconstructed $FPCT$ of a tree is the PCT of that tree. Fig. 3.14 shows an example where this property holds.

Property 3.16 (Reconstructed $pFPCT$). $\psi(FPCT(t)) = PCT(t)$

If a $pFPCT$ is a subset of the $FPCT$ of tree t , i.e. the itemset representation of tree t , its reconstructed tree is a subtree of t .

Property 3.17. Let TDB be a tree database, then

$t \in TDB$ and $pFPCT \in FPCT(t)$ if and only if $\psi(pFPCT) \preceq_e t$.

Proof. From Property 3.16: $\psi(FPCT(t)) = PCT(t)$ and $pFPCT \in FPCT(t)$, we have $\psi(pFPCT) \preceq_e PCT(t)$. It is known that $PCT(t)$ is the position-constrained top-left mirror of t on the DSM of TDB , thus $\psi(pFPCT) \preceq_e t$. ■

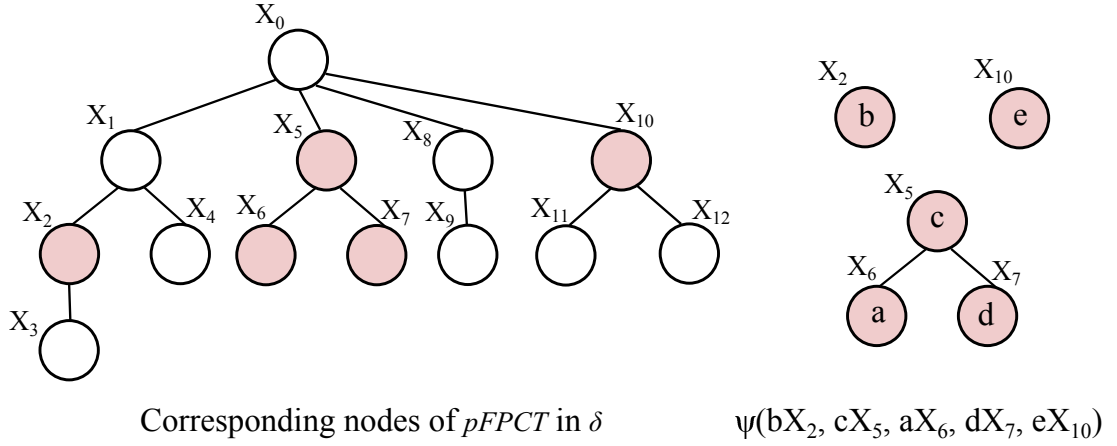


Figure 3.13: The reconstruction of $pFPCT = \{bX_2, cX_5, aX_6, dX_7, eX_{10}\}$.

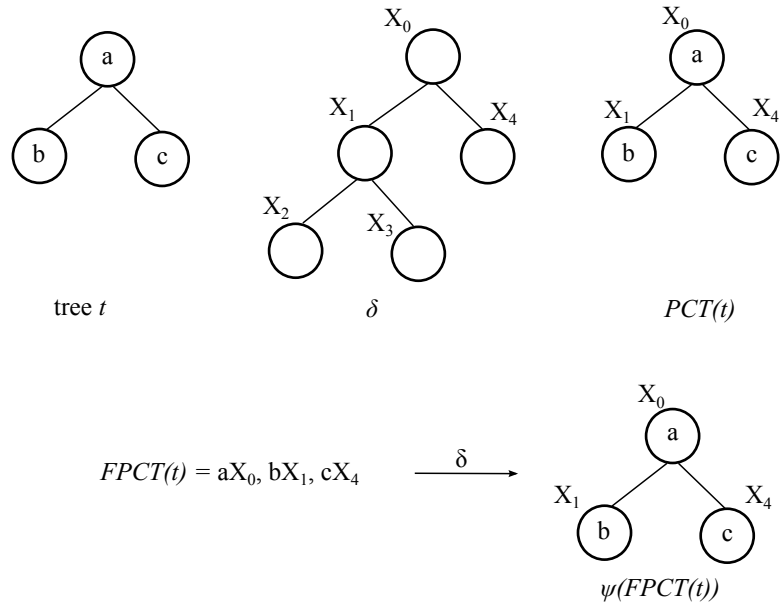


Figure 3.14: An example of Property 3.16.

Property 3.18. Given TDB is a tree database, and a $pFPCT$ being frequent in the database of $FPCT(t)$ with $t \in TDB$ (i.e. the itemset representation of TDB), the reconstructed tree of $pFPCT$ ($\psi(pFPCT)$) based on the DSM of TDB is also frequent.

After the characteristics of $pFPCT$ are defined, the algorithm that converts a $pFPCT$ to tree representation can be described. The pseudo-code of the algorithm is given in Listing 5.

3.2.4 Illustration of the Main Concepts

The method and concepts defined in previous sections are summarised and illustrated within an example, where the structure-preserving tree mining approach is used to discover frequent embedded subtrees from the tree database T_{EX} . Fig. 3.16 shows the

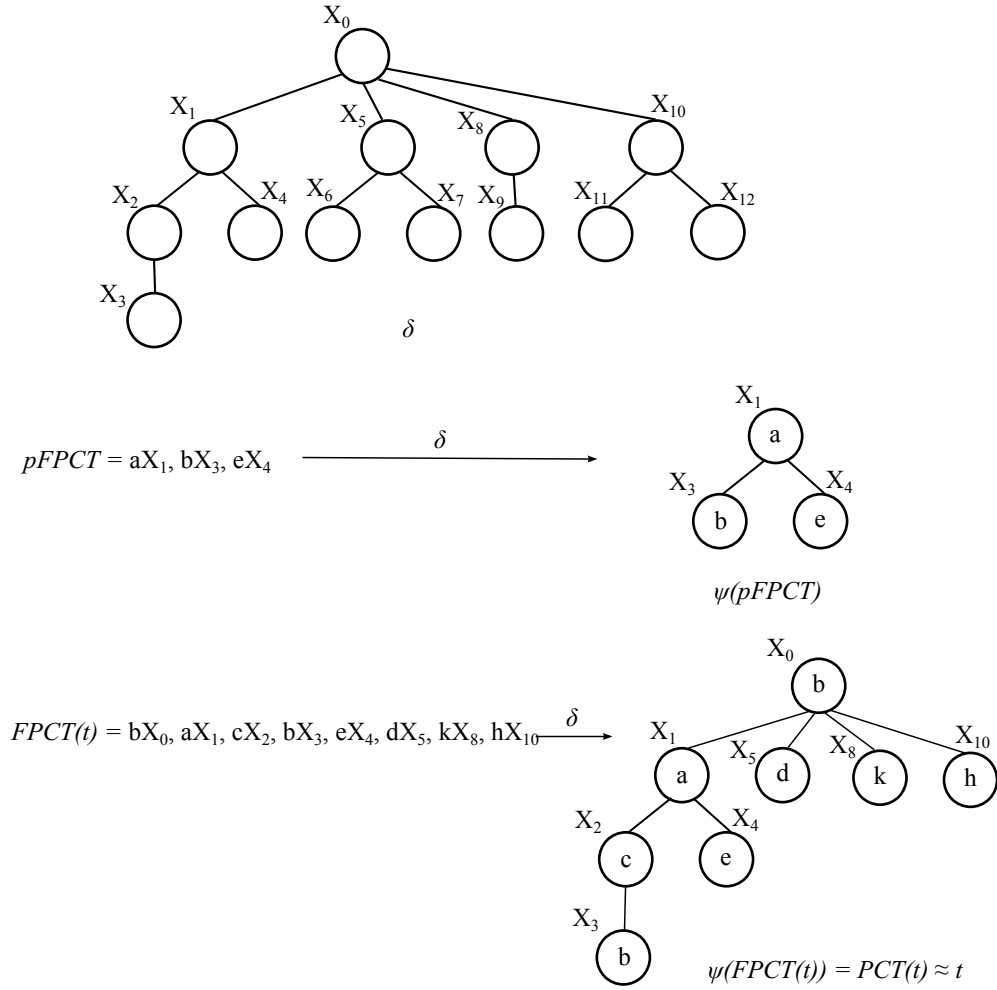


Figure 3.15: An example of Property 3.17.

five main steps of the process.

Firstly, a *DSM*, denoted by δ , is extracted from the database. Secondly, the top-left mirror of each tree instance on the *DSM* (*PCT*) is identified. Thirdly, the *FPCT* representation of each tree instance is extracted from the *PCT*. *FPCT* was previously called as the itemset format of tree instances. Finally, a frequent itemset mining algorithm is applied to the set of *FPCT*s. The resulting patterns are called *pFPCT*s. In the figure, only two *pFPCT*s are given as an example. In the final step, tree-like patterns are reconstructed from the *pFPCT*s. The reconstructed tree of a *pFPCT* is denoted as $\psi(pFPCT)$.

3.2.5 DSM with Minimum Support

The number of nodes of a *DSM* can be very large because it has to be large enough to match with all structural varieties of all instances in database. To reduce the structural complexity, nodes that are (top-left) mapped by less than a certain number (i.e. minimum support) of tree instances in the database are removed from the *DSM*.

Algorithm 5 *Reconstructing subtrees from a pFPCT.*

Input: $pFPCT = p$, the DSM tree (δ)

Output: $\psi(pFPCT)$ in pre-order string encoding

```

1: for each  $m \in p$  do
2:   Extract position  $X_i$  from  $m$  and add to  $positionSet$ 
3: for each  $pos \in positionSet$  do
4:   if  $pos = \delta.root$  and  $\delta.root.visited = false$  then  $\triangleright \delta.root$  is the root of  $\delta$   
and  $\delta.root$  is not visited
5:      $\delta.root.visited = true$ 
6:     ADD node value at position  $pos$  to  $\psi(pFPCT)$ 
7:   for each child node  $c \in \delta$  do
8:      $Algorithm3(c, p)$ 
9:   for each  $pos \in positionSet$  do
10:    if  $pos = \delta.root$  then
11:      for each  $a$  is the ancestor of  $\delta$  do
12:        if  $a.root.visited = true$  then
13:          ADD ‘-1’ to  $\psi(pFPCT)$ 
    
```

Definition 3.19 (DSM with a minimum support). *The DSM with a minimum support of m of a tree database TDB is denoted as δ_{TDB}^m , or δ^m . δ^m is a tree $(V_{\delta^m}, V_{\delta^m0}, E_{\delta^m})$, such that*

- $\forall t \in TDB, t \preceq_{tl} \delta^m$;
- $\forall x \in V_{\delta^m}, \exists t = (V_t, V_{t0}, E_t) \in TDB, \exists m.u \in V_t : \zeta_t(u) = x$. Note that ζ_t is the top-left mapping function of t on δ^m .

In practice, relative minimum support is more commonly used than (absolute) minimum support. A DSM with a relative minimum support of n is defined as $\delta_{TDB}^{(n)} = \delta_{TDB}^{n|TDB|}$. For instance, the DSM of the database T_{EX} with a relative minimum support of 40%, denoted as $\delta_{T_{EX}}^{(0.4)}$, or $\delta^{(0.4)}$, is equivalent to the DSM with a minimum support of 2, denoted as $\delta_{T_{EX}}^2$. $\delta^{(0.4)}$ is shown in Fig. 3.17(a). The position-constrained top-left subtrees of t_i with $i \in [0..4]$ on $\delta^{(0.4)}$ are shown from Fig. 3.17(b) to Fig. 3.17(f). Note that several tree nodes can no longer be mapped to the reduced DSM, such as nodes b and c of t_2 , nodes b, c , and e of t_3 , and nodes b of t_4 .

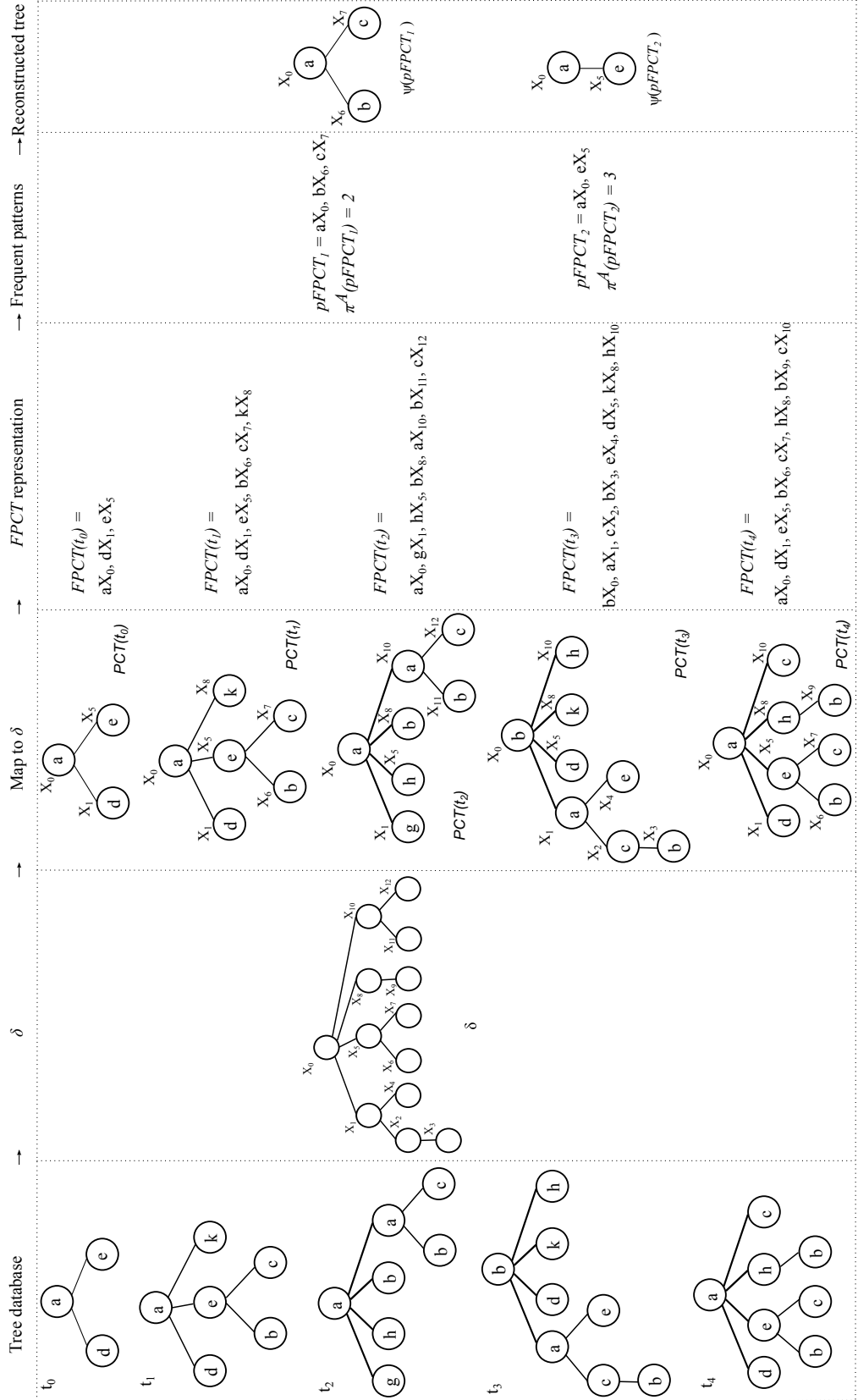


Figure 3.16: Illustration of the main concepts in the structure-preserving tree mining approach.

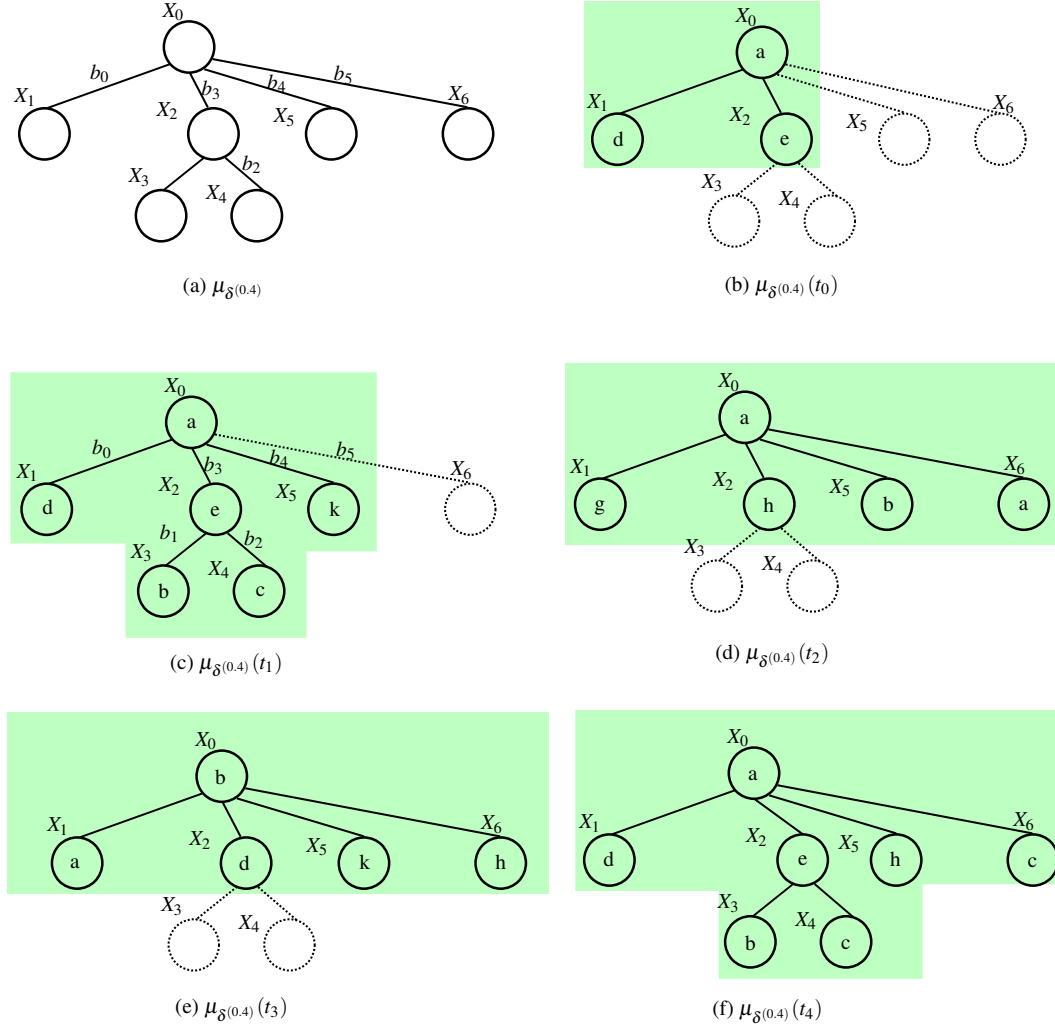


Figure 3.17: The *DSM* of T_{EX} with a support threshold of 40% is shown in (a), and the mapping of tree instances of T_{EX} to the *DSM* are shown in (b), (c), (d), (e), and (f).

3.3 Time Performance of *DSM*-based Frequent Subtree Mining and Traditional Methods

In this section, the whole hospital process log was examined and the closed subtree mining task was selected for the benchmarking purpose. The structural properties of the hospital dataset (van Dongen, 2011) are described as follows: $|\text{transactions}| = 1143$; average encoding length (average number of nodes in pre-order encoding) = 2134.5; maximum tree size (maximum number of nodes in a tree) = 14533; average tree height = 2; average tree fan-out = 8.27 (average number of children); average tree size = 1067.7; maximum tree height = 2; maximum tree fan-out = 1834. The reason that the average tree height and maximum tree height are both equal to 2 is because this process log have multiple traces, each of which contains multiple events, and each of which contains multiple attributes. The number of tree nodes is large because there is

a large number of event nodes.

Closed subtree mining is considered as one of the state-of-the-art techniques in the *FSM* field, and the *CMTreeMiner* and *DryadeParent* (Termier et al., 2008) algorithms were selected. Note that the *DryadeParent* algorithm assumes attribute trees. An attribute tree is a tree where all sibling nodes of any certain node must be different. Because the tree representation of the hospital dataset is not a valid attribute tree database, each *event* node in each tree was renamed so that each of them is assigned a different identifier. The resulting attribute tree database with all identical sibling nodes relabelled is called *SbRI*. The structural properties of the *SbRI* dataset are similar to that of the hospital dataset except that the number of labels is larger due to the node relabelling.

The position-constrained frequent subtree mining method was used in conjunction with a frequent closed itemset mining algorithm *LCM* (Uno et al., 2004). This integrated method is called *DSM-LCM*. The *CMTreeMiner* and *DSM-LCM* were tested on *SbRI* data to observe the effect of the sibling relabelling on the time performance. A closed sequential pattern mining method, *Clospan* (Yan et al., 2003), was also selected as another subject for the comparison since sequence mining had been used before to assist in process model discovery (van der Aalst, 2011a).

The time performance of different approaches is shown in Fig. 3.18. Note that each algorithm runs on the original and the *SbRI* data. In the original hospital dataset, the results of *CMTreeMiner* are not seen on the chart since it could not complete the task for any given minimum support. This is probably due to the high complexity of the dataset. *Clospan* gives a slightly better result since it can still produce results down to the support value of 70%. The *DSM-LCM* method performs better than others as the time to finish the task is almost similar across the support ranges—down to 40% with a relatively large increase at 30%.

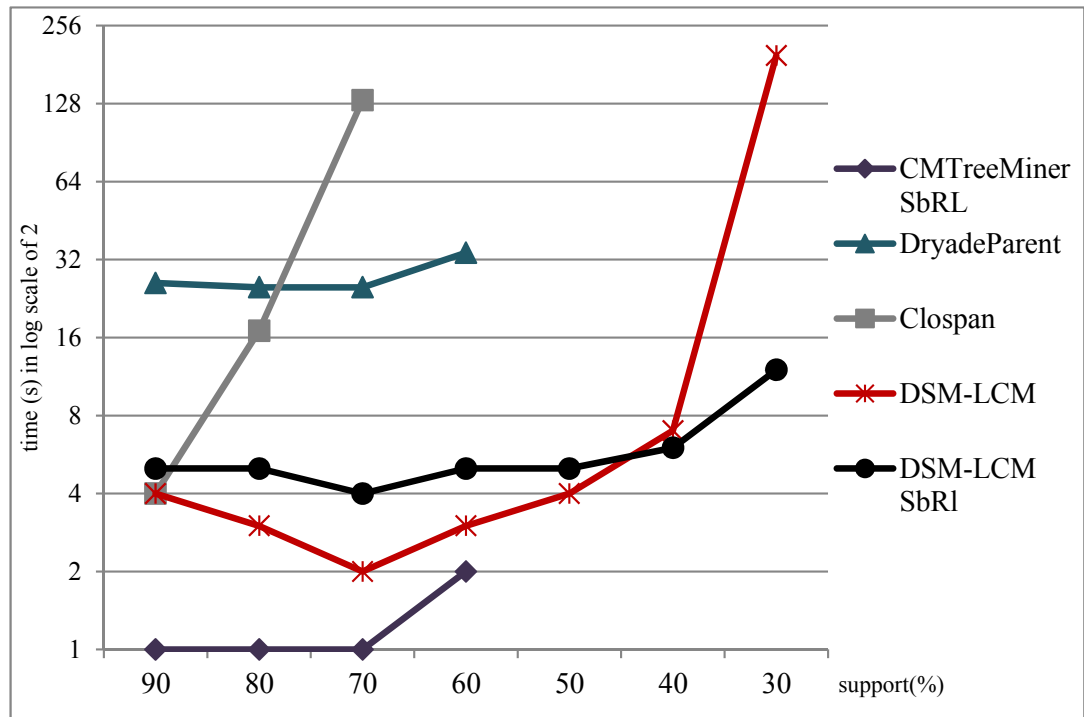


Figure 3.18: Time performance of different methods.

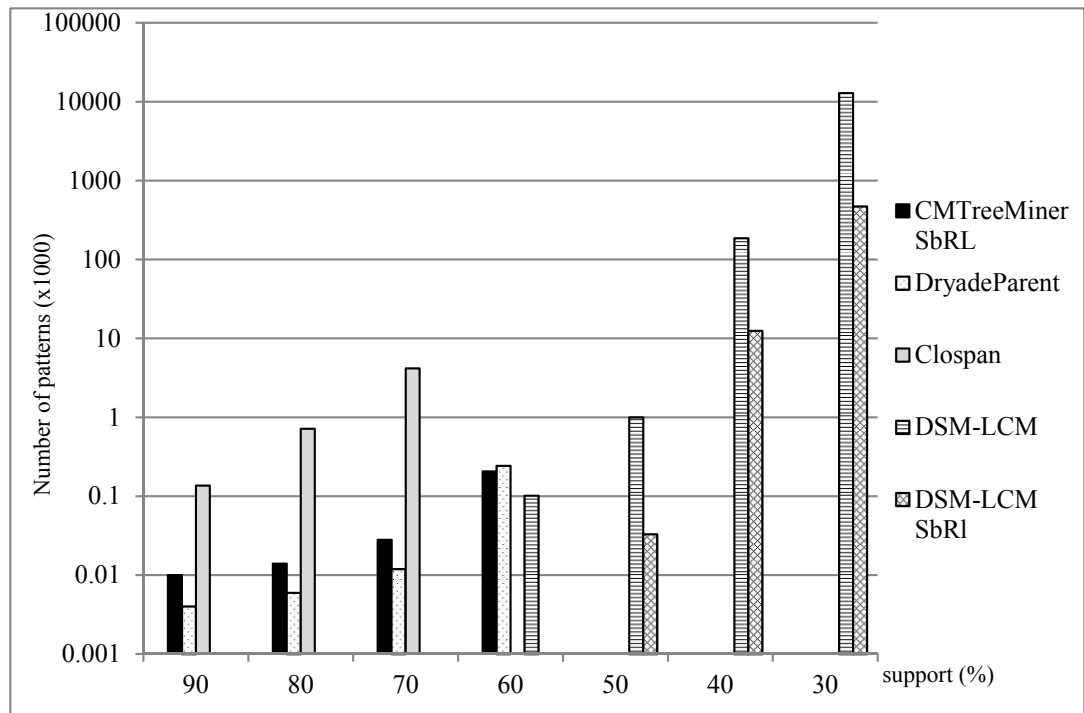


Figure 3.19: Number of patterns of different methods.

In the *SbRI* data, *CMTreeMiner* performs much better than it does on the original dataset. This can be explained by the high number of node repetitions, e.g. *event* in the original data. When these nodes are relabelled, the *CMTreeMiner*'s performance is comparable to that of *DryadeParent*. The best performance for this dataset is achieved by the *DSM-LCM* method. There is not much time difference between the *DSM-LCM* methods when applied to the two datasets, except at the support threshold of 30%.

The number of frequent patterns found by different methods is shown in Fig. 3.19. It can be seen that the lower the support, the more patterns can be detected, and the number of frequent patterns found by the *DSM*-based method is comparable to that of *DryadeParent*. The *DSM-LCM* method gives only one pattern at higher support value due to the position constraints. When the minimum support drops to 60%, the *DSM*-based method discovers similar number of patterns as other approaches. At even lower support thresholds, the method increasingly discovers more patterns, thus overall finding more frequent associations from this data.

In parallel to this work, the *DSM*-based frequent subtree mining method is evaluated on a variety of synthetic and real world datasets (Hadzic et al., 2015). The results demonstrate that the *DSM*-based method provides superior performance over state-of-the-art frequent subtree mining methods, such as *DryadeParent* and *CMTreeMiner*, in a number of area, e.g. it can handle lowering minimum support threshold in complex datasets (database with many deep and/or wide trees), and the position-constrained subtree patterns are more informative than traditional subtree patterns.

3.4 The Proposed Methods

As discussed in Section 3.1, the tree-structured process log is often characterised by the repetition of nodes within a trace. In addition, the number of events in an event log can be quite large. Such two factors could lead to performance problems for traditional frequent subtree mining approaches. As claimed in a number of works (Bui et al., 2012a,b; Hadzic et al., 2015), the structure-preserving tree mining approach does not only increase the number of techniques that are directly applicable to event data, but also reduces the complexity of the analysis for which common *XML* mining techniques based on frequent subtree extraction have shown to be infeasible. The time performance evaluation presented in Section 3.3 confirms this claim.

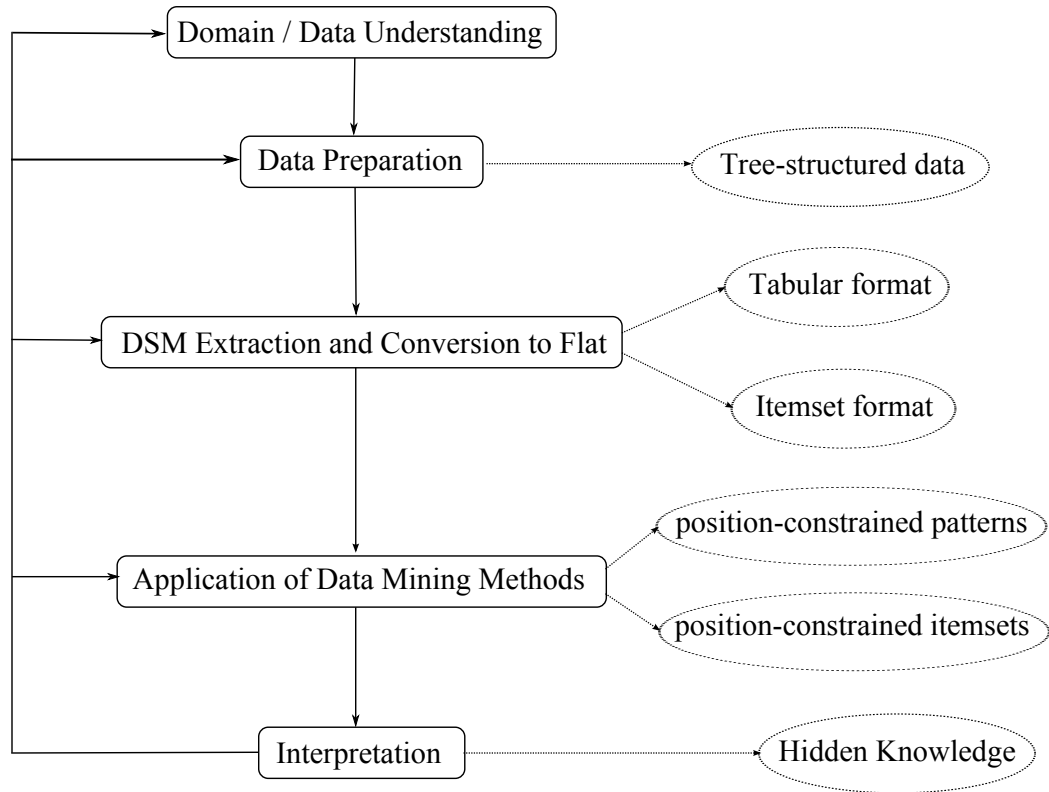
Another useful property of the position-constrained frequent subtree mining approach is that the exact position of nodes/attributes in the patterns is indicated. This property could be useful in process mining as events/actions are distinguished based on their exact occurrences within a trace of events.

Due to the above reasons, the structure-preserving tree mining approach is selected as the basis for our proposed process log analysis method, which is described in Sec-

tion 3.4.1. In addition, in Section 3.4.2, an exploratory process log analysis method is proposed. This method can be used when there is a lack of domain knowledge to guide the analysis.

3.4.1 The Position-constrained Tree-structured Mining Approach for Process Log Analysis (*PCFSM*)

The *PCFSM* method is divided into five main phases, which are presented in Fig. 3.20. The first phase starts with the acquisition of general knowledge from the domain and identifying goals of the process log analysis (domain understanding). The next step (data understanding) is to identify how relevant domain knowledge is represented in the dataset. In the data preparation phase, the process log originally in *XML/MXML/XES* format is converted into the tree-structured representation. In the next phase, the *DSM* is extracted from the database, and based on this structure the event log is transformed to a flat representation, which enables the direct application of well-established data mining techniques. Depending on the user's goals, the data mining phase may employ certain data mining methods such as association rule learning, decision tree learning, clustering, etc. As an example, process instances can be labelled according to different business requirements and then are used to train a classifier for the prediction purpose. In the interpretation phase, the results obtained from previous phases should be analysed and interpreted in a way that is understandable and actionable by the domain experts. The details of each phase are described as follows.

Figure 3.20: The *PCFSM* method.

Domain/Data Understanding

This phase is essential in any knowledge discovery in databases (*KDD*) process. Almost all *KDD* process models start with the task of getting to know the domain and identifying user goals (Kurgan and Musilek, 2006). For instance, for the hospital process log described in (Bose and Aalst, 2012), the analyst should know that diagnosis affects treatment procedures, so that treatment process instances relating to different diagnoses are considered separately during the analysis. Furthermore, the analyst has to know how the diagnosis or the treatment is represented in the data in order to retrieve them correctly and/or combine related data together for further processing. Without any basic knowledge of the domain/data or a domain-expert to provide feedback and guidance during different stages, it is difficult to satisfy the goals of the analysis.

Data Preparation

Depending on the nature/quality of the data or specific purpose of the process mining task, different pre-processing techniques could be used, e.g. data cleansing (noise, missing value, inconsistency handling), identifiers/repetitions removal, discretisation, feature selection. If the *MXML/XES* format of data is not available at the beginning of this phase, the *extract transform and load* (ETL) methods can be used (van der Aalst, 2011a). Furthermore, since in the proposed approach the aim is to capture the

Table 3.4: Mapping node labels to integers.

Text	ID
trace	0
AgeGroup:21-65	1
Consultation	2
Administration	3
BloodTest	4
RadioTherapy	5
GeneralLab	6
AgeGroup:>65	7
CytologicEvaluation	8
ObstetricsClinic	9
Pathology	10
AgeGroup:<21	11
PhoneConsultation	12
CTAbdomen	13
MRIAbdomen	14
MicroscopicExamination	15
Radiology	16
Microbiology	17
FlamePhotometers	18

exact position of an attribute/event in the patterns extracted from process instances, the attributes should be sorted in the same order across trace or event nodes. The *MXML/XES* data is then modelled as a set of rooted ordered labelled trees and represented in a pre-order string encoding. Note that, *XML* elements, attributes and their values are not separated into separate nodes. Therefore, a node label is a representative of both the element and the element value, while adhering to the hierarchical properties of the document.

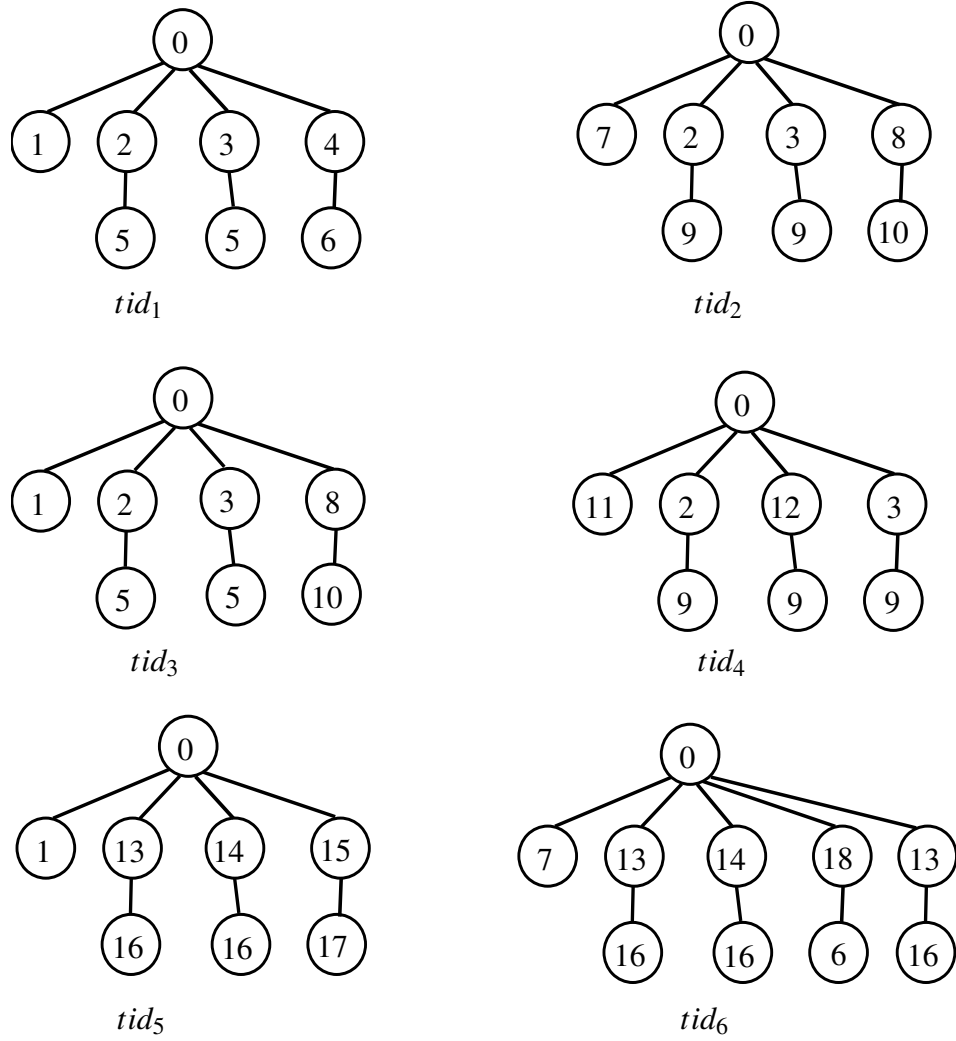
DSM Extraction and Conversion to Flat Representation

This step was described in detail in Section 3.2.1 and 3.2.2. To improve the speed of our approach, all text values in nodes are mapped into integers. For the tree database T_{HOS} , the mapping table is displayed in Table 3.4. The resulting database with text values replaced by integer values is presented in Fig. 3.21.

The *DSM* of the tree database T_{HOS} is shown in Fig. 3.22. The *FDT* and its itemset (*FPCT*) representation are shown in Table 3.5 and Table 3.6, respectively.

Application of Data Mining Methods

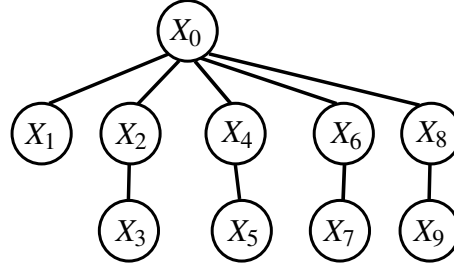
The *FDT* and the itemset representation (*FPCT*) enable the direct application of data mining methods originally developed for vectorial data to tree structured process data.

Figure 3.21: T_{HOS} with integer values.

In particular, the *FDT* can be used for clustering/classification, and the itemset representation can be used for frequent pattern mining/associative classification. The discovered position-constrained subtree patterns in itemset format are then reconstructed to tree structured format using the *DSM* as a reference model.

Applying a frequent itemset mining algorithm to the database in Table 3.6 would result in many frequent itemsets. At a minimum support of 2, an example of a frequent itemset found is $\{0X_0, 1X_1, 2X_2, 5X_3, 3X_4, 5X_5\}$, which can be translated to $\{\text{'trace'}X_0, \text{'AgeGroup=21-65'}X_1, \text{'Consultation'}X_2, \text{'RadioTherapy'}X_3, \text{'Administration'}X_4, \text{'RadioTherapy'}X_5\}$. This frequent itemset is then reconstructed to a tree format, which is shown in Fig. 3.23.

Comparing the position-constrained subtree pattern (Fig. 3.23) with its corresponding traditional subtree (Fig. 3.3(a)), one can see that using the *PCFSM* method, the positional information of each node in the subtree pattern is indicated. In process mining, this translates to an exact occurrence of an event (or any other aspect of the process) within a trace. This characteristic of distinguishing subtrees based upon their exact

Figure 3.22: The DSM of T_{HOS} .Table 3.5: The FDT of T_{HOS} .

X_0	X_1	b_0	X_2	X_3	b_1	b_2	X_4	X_5	b_3	b_4	X_6	X_7	b_5	b_6	X_8	X_9	b_7	b_8
0	1	1	2	5	1	1	3	5	1	1	4	6	1	1	0	0	0	0
0	7	1	2	9	1	1	3	9	1	1	8	10	1	1	0	0	0	0
0	1	1	2	5	1	1	3	5	1	1	8	10	1	1	0	0	0	0
0	11	1	2	9	1	1	12	9	1	1	3	9	1	1	0	0	0	0
0	1	1	13	16	1	1	14	16	1	1	15	17	1	1	0	0	0	0
0	7	1	13	16	1	1	14	16	1	1	18	6	1	1	13	16	0	0

occurrences would cause the $PCFSM$ method not to detect the subtree displayed in Fig. 3.3(b) at the support threshold of 2. This is because the right hand side of the subtree (i.e. node *Administration* and *ObstetricsClinic*), occurs at different positions within process instance 2 and 4 of Fig. 3.1.

The $PCFSM$ method would not consider groups of events as similar if additional/different events occur within the group. For example, in process instance 4 of Fig. 3.1, the activity *PhoneConsultation* occurs in between the activity *Consultation* and *Administration*. This is important as the process analyst can be certain that the position-constrained subtree pattern reflects exact similarity of groups of events across traces, where no additional or different events occur in between.

Interpretation

In this phase, the patterns are evaluated for their specific use in a given application. For example, in outlier/exception detection, the low-occurring frequent subtree pat-

Table 3.6: The itemset format of T_{HOS} .

T_{HOS}	Itemsets
tid_0	$0X_0, 1X_1, 2X_2, 5X_3, 3X_4, 5X_5, 4X_6, 6X_7$
tid_1	$0X_0, 7X_1, 2X_2, 9X_3, 3X_4, 9X_5, 8X_6, 10X_7$
tid_2	$0X_0, 1X_1, 2X_2, 5X_3, 3X_4, 5X_5, 8X_6, 10X_7$
tid_3	$0X_0, 11X_1, 2X_2, 9X_3, 12X_4, 9X_5, 3X_6, 9X_7$
tid_4	$0X_0, 1X_1, 13X_2, 16X_3, 14X_4, 16X_5, 15X_6, 17X_7$
tid_5	$0X_0, 7X_1, 13X_2, 16X_3, 14X_4, 16X_5, 18X_6, 6X_7, 13X_8, 16X_9$

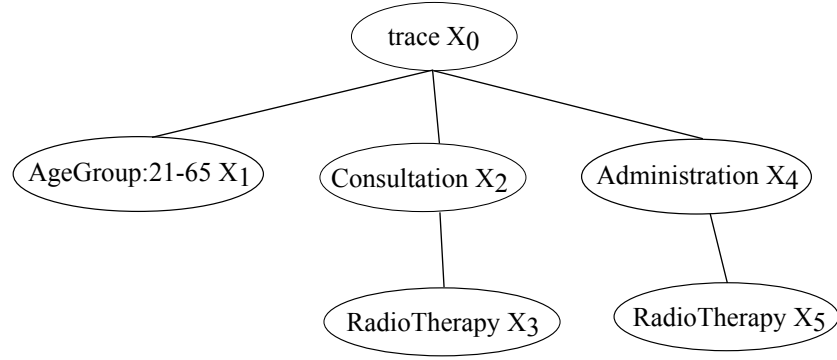


Figure 3.23: A reconstructed position-constrained pattern of T_{HOS} .

terns might indicate characteristics of outlying or exceptional cases. The difference between these and the more frequently occurring patterns reflecting the norm will be investigated. Once the outlying instances are detected, they will be labelled as outlying and others as norm. A classification model can then be used to predict the outlying behaviour when a set of preconditions during a business process execution path become true.

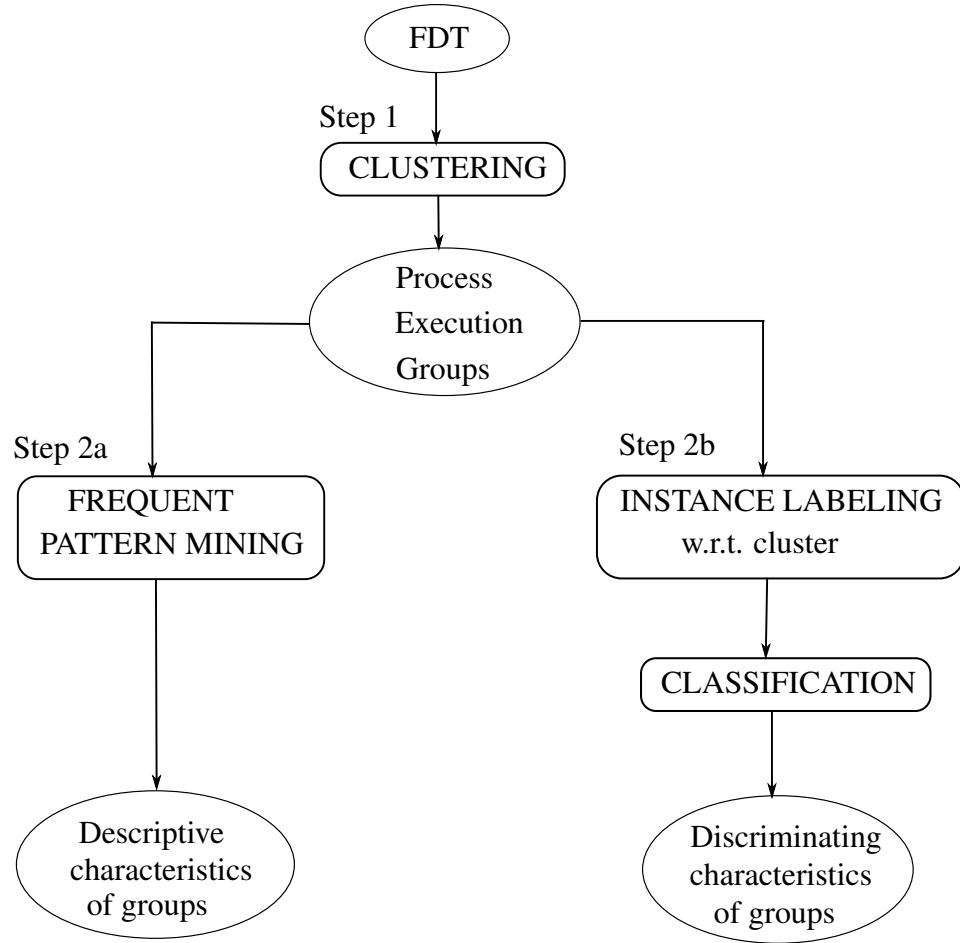
In the context of classification, the process log can be labelled with respect to a need to learn more about a particular business process aspect (e.g. duration, performance bottleneck, and known cases of fraud). Classification based on *PCFSM* will be demonstrated in the next chapter.

3.4.2 Exploratory Process Log Analysis Method

Based on the *PCFSM* method, an Exploratory Process Log Analysis *Exploratory Process Log Analysis (EPLA)* method is proposed. This method is recommended when the process analysts do not have a clear goal in mind, or they want to explore the data in the most generic way. The main steps of *EPLA* are presented in Fig. 3.24. In the first step, the process log is clustered to groups of process instances which have similar execution paths. A frequent subtree mining method is then used to discover the descriptive characteristics (common subtrees among instances) of each cluster (step 2a). To detect discriminating characteristics of the clusters, similar process instances are grouped together and assigned a virtual cluster label. Next, a classification algorithm is used to discover the distinguishing characteristics of each group (step 2b).

The suggested method has an exploratory nature, where unanticipated groups of processes and differences among process executions might be detected. Hence, the method is suitable when the domain knowledge is unavailable to guide the discovery process.

A useful property of this method is that it does not restrict the clustering algorithm to be used. In fact, the users can choose either *PCFSM*-based methods, or any other

Figure 3.24: The *EPLA* method.

XML mining methods, such as *XProj* (Aggarwal et al., 2007), *XRules* (Zaki and Aggarwal, 2006), and *Dalamagas's* method (Dalamagas et al., 2005), for the exploratory analysis. Please note that in (Hadzic et al., 2011a), a *DSM*-based clustering approach was evaluated against traditional *XML* clustering methods in synthetic process logs. The structural complexity and large size of process data resulted in poor time performance for the competing *XML* clustering methods, while the *DSM*-based clustering method arrived at high quality clusters in a significantly shorter time.

3.5 Discussion

The proposed method is adapted from a standardized knowledge discovery process suggested by (Fayyad et al., 1996). Compared to his nine-step process, our approach has some differences. The two models are quite similar except that the last step (knowledge dissemination) of his model is considered out of this scope in this work and our approach has an additional step before the data mining step called *DSM* extraction. Our second phase (data preparation) is a combination of step two (data selection), three (data cleaning) and four (data reduction or transformation) in Fayyad's model.

The third phase in our approach (data mining) is roughly similar to step five (choosing data mining task), six (choosing data mining algorithm) and seven (executing the algorithm) in his model.

It is worth noting that our approach is iterative where at each phase there is a loop back to previous phases. This is a common characteristic of all *KDD* process since its nature is exploratory where mistakes or uninteresting discoveries frequently happen. Since our tree-structured process log analysis approach is based on a specific variation of tree mining, one can adapt the approach to use other tree mining techniques in the data mining phase without the need of the *DSM* extraction phase. For example, one can directly apply the *FSM* algorithm *CMTreeMiner* (Chi et al., 2004c) to the pre-processed *XML* process log to reveal frequent associations among group of activities and their attributes (this method does not take into account the exact position of each subtree pattern as illustrated in Section 3.1.1). One can also apply the associative subtree classifier *XRules* (Zaki and Aggarwal, 2006) to build prediction model for future process instances. Tree-based clustering techniques like *XRep* (Costa et al., 2004) can be used in this phase to detect group of similar process instances.

The advantage of our approach is that the tree-structured data is transformed into a presentation that is suitable for the application of traditional data mining methods without the loss of temporal, positional and contextual information which could be useful for process log analysis tasks. In case where traditional subtrees are needed, the position information can be removed from the flat data format during the subtree reconstruction process. Moreover, the *PCFSM* method could reduce the structural complexity often faced by traditional approaches when searching for frequent patterns in process logs, thanks to the structure-preserving conversion method.

While a number of techniques are made applicable within the proposed approach, the choice and sequence of their use is dependent on the goals of analysis. For example, if there is an aspect of process logs for which causal factors are to be determined, classification methods can be used on process instances labelled according to that aspect. If the event data are large or heterogeneous, clustering methods are often used to obtain more manageable data segments and obtain more focused results in subsequent analysis.

Our proposed method differs from the process mining frameworks discussed in Section 2.2 in a number of ways. First, unlike the studies of (Grigori et al., 2004) and (Bozkaya et al., 2009), which provides a general architecture for mining event logs, our work focused on the development of a process mining method that is effective, extensible and scalable. The framework proposed by (De Weerd et al., 2013) analyses specific attributes of the data instead of treating data as a whole as in our method. In (Rebuge and Ferreira, 2012), the suggested framework is based on the control-flow analysis of event data, whereas our technique does not require process model at all.

Overall, this work extends the available pool of process analysis techniques and allows efficient knowledge discovery from process logs in a more direct and unbiased manner.

Chapter 4

Evaluation of PCFSM on Process Logs

In this chapter, extensive experimental evaluation on publicly available real-world and synthetic datasets highlights the benefits of the *PCFSM* method with respect to classification capabilities and exploratory analysis. Different data mining techniques such as clustering, decision tree learning, frequent pattern mining and associative classification are used in the experiments.

The experimental settings and data descriptions are discussed in Section 4.1. Next, the evaluations of the exploratory analysis method and the *PCFSM* method on the hospital dataset are presented in Section 4.2. In Section 4.3, classification methods based on *PCFSM* are evaluated on labelled datasets where the purpose is to identify the properties of process instances that characterise a particular business goal. Finally, Section 4.4 discusses the lessons learned from the experimentation.

4.1 Experimental Settings

From the experiments, three benefits of the *PCFSM* method are demonstrated including (i) distinguishing knowledge patterns based on the exact occurrence of the activities within a process, (ii) enabling the direct application of a wide range of data mining techniques on tree-structured process logs, and (3) the analysis does not require a process model. Please note that the proposed method was not compared to traditional process mining algorithms because the results of the analysis are incompatible.

All experiments in this thesis were conducted on a Linux machine, Intel Xeon E5345 at 2.33 GHz, 8 GB RAM and 4MB Cache Open SUSE 10.2 64bit. The datasets used for the experiments are described in the following sections.

4.1.1 Hospital Dataset

The hospital dataset is a real-world data set containing patient treatment processes from the *Gynaecology* department of a hospital (van Dongen, 2011; Bose and Aalst,

2012). The data is in *XES* format. At the top level, each patient treatment process is represented by a tag *trace*. Then, each *trace* contains a set of attributes, such as *diagnosis*, *diagnosis code*, *treatment*, *treatment code*, *start time*, and multiple *event* tags, each of which consists of a set of nine attributes, such as *org:group* (the department in which the activity occurred), *concept:name* (the name of the activity), *time:timestamp*. Attributes such as *specialism code*, *treatment code combination ID*, are removed from the log because they are either unique for each trace or contained elsewhere in the trace. Event attributes such as *lifecycle:transition* and *number of executions* are also removed as their values are the same across all instances.

The dataset describes a large variety of processes characterising different treatments/diagnosis, which themselves are overlapping. These properties of the dataset were described in (Bose and Aalst, 2012), with a clear indication that one needs to focus on only one aspect/segment of the data, such as a single diagnosis or treatment code. Hence, in this experiment, the set of processes where the describing attribute is a single diagnosis code are focused. A subset of data that contains traces of patients who are diagnosed with the code *M13*—the most frequent diagnosis code in the dataset—is selected for analysis. If a trace has multiple treatment codes, they are concatenated into a single treatment code, while preserving the order of the concatenation across records with the same treatment code combination. Binning techniques (Han and Kamber, 2006) are used for attributes such as "Age" and "TraceDuration". The values of the "Age" attribute is discretised to three bins, i.e. "Young" (age ≤ 20), "Working" (age ≤ 65 and > 20) and "Retired" (age > 65). The values of the "TraceDuration" attribute is discretised to 14 bins with values ranging from 0 to 1500 days.

4.1.2 Teleclaim Dataset

The synthetic teleclaim process log describes the handling of claims in an insurance company (van der Aalst, 2011b). The log contains 46138 events related to 3512 cases (claims). One typical process is that the customers *file the claims*, the centre *checks information*, *registers to the system*, the claim is then *quickly-checked* by a claim handler, after that it is fully *examined*; the officer *advises the claimant* and starts *payment*; finally the claim is *closed*. The purpose of this experiment is to build a classification model to identify four possible outcomes of a claim such as *processed*, *rejected*, *insufficient information*, and *not liable*. Each trace is labelled from one of the four values as described above.

4.1.3 Telephone Repair Dataset

This dataset describes an artificial log of telephone repair process (van der Aalst, 2011c). One example of process instance starts by a customer *registering a telephone*

Table 4.1: Structural characteristics of data sets.

Dataset	Tr	Avg L	Avg D	Avg F	Avg T	Max D	Max F	Max T
Hospital	252	934.5	2	7.4	467.7	2	352	2796
Insurance Claim	3512	114.1	2	4.0	57.5	2	73	338
Telephone Repair	1104	106.0	2	4.5	53.5	2	27	112

for repair; the telephone is *analysed*; then *transferred* to either *simple repair* team or *complex repair* team; at the same time the customer is *informed* of the condition of their device; once the telephone is repaired it is *tested*; if not fixed it is then *sent back* for repair; the case is *archived* after the telephone is fixed.

4.1.4 Summary on Datasets

The structural properties of the datasets used in this chapter are presented in Table 4.1. Note that $|Tr|$ = number of transactions, $Avg(|L|)$ = average length of pre-order string encoding, $Max(|T|)$ = maximum size of trees, $Avg(|D|)$ = average height of trees, $Avg(|F|)$ = average fan-out of trees, $Avg(|T|)$ = average size of trees, $Max(|D|)$ = maximum height of trees, $Max(|F|)$ = maximum fan-out of trees.

4.2 Evaluation of the *EPLA* method

4.2.1 Discovering Process Execution Groups

The hospital dataset is clustered into different process execution groups using the *PCFSM*-based clustering technique similar to the one proposed in (Hadzic et al., 2011a). The tree-structured data is converted to *FDT* representation and the *CLUTO* clustering tool-kit (Karypis, 2003) is then used to form clusters. Note that evaluations of the clustering method based on position-constrained subtrees against state-of-the-art *XML* clustering methods on data of varied complexity, including a complex synthetic event log, were already provided in (Hadzic et al., 2011a). The number of clusters (clustering parameter k) is trialled with different values and *Euclidean*, *Jaccard* and correlation distance measures are used. Table 4.2 shows the average internal similarity and external similarity (abbreviated as *ISim* and *ESim*, respectively) values of all clustering solutions with the latter shown in parentheses. A clustering solution that has a relatively small number of clusters while still having the large gap between the average of *ISim* and *ESim* is selected for further analysis. When $k=4$ and *Euclidean* distance measure is used, the algorithm gives us six clusters with a relatively large gap of average *ISim* and *ESim* (0.551). The size and the *ISim* and *ESim* of each cluster are presented in Table 4.3. It can be seen that the top two clusters outperform the remaining clusters in terms of the difference between the *ISim* and *ESim*. Each cluster

Table 4.2: Average internal and external similarity for different clustering options.

k	Correlation	Euclidean	Jaccard
3	0.517(0.153)	0.388(0.001)	0.306(0.000)
4	0.574(0.197)	0.552(0.001)	0.287(0.001)
5	0.586(0.209)	0.467(0.012)	0.452(0.001)
6	0.594(0.213)	0.573(0.010)	0.410(0.002)
7	0.610(0.223)	0.607(0.028)	0.471(0.006)
9	0.653(0.250)	0.637(0.040)	0.500(0.002)

Table 4.3: A clustering result at $k = 4$.

Cluster	Size	ISim	ESim
0	13	0.999	0.003
1	28	0.880	0.002
2	58	0.552	0.001
3	5	0.488	0.000
4	37	0.200	0.001
5	74	0.193	0.000

contains process instances that share similar characteristics and a frequent pattern mining algorithm can be used to discover the characteristics that are prevalent among the instances.

Since the main objective of *EPLA* method is help process analysts or domain experts explore the process log in a meaningful way, and no ground-truth checking is required. From the instances of the top three clusters, it was observed that cluster 1 (13 instances) includes patients that are in the “retired” age group, cluster 2 (28 instances) includes patients that are in the “working” age group, having treatment duration normally lasting from one to one and a half year, and cluster 3 (58 instances) consists of patients that are also in the “working” age group, but having treatment duration either very short (less than half a year) or very long (more than two years). This observation could help process analysts to narrow down the scope of data that they want to analyse.

4.2.2 Discovering Descriptive Characteristics of Groups

Two methods were used to discover common characteristics of a group of instances: (i) applying a frequent closed subtree mining algorithm *CMTreeMiner* to the tree-structured data, (ii) applying a frequent closed itemset mining algorithm *LCM* (Uno et al., 2004) to the tabular format (*FDT* representation) of the two clusters (abbreviated as *DSM-LCM*). The minimum support threshold was set at 90% for both approaches. Since the aim was to discover the subtree patterns that reflect the characteristics of the majority of process execution instances within a group, a rather large support threshold

was used. For other application aims, one would set lower support thresholds, which typically result in more specific (but larger) patterns characterising smaller subsets of process executions.

The subtrees of Fig. 4.1 and Fig. 4.2 indicate some distinguishing characteristics of clusters 0 and 1, respectively. For example, cluster 0 is characterised by 2 events, each having a producer code of *SGNA* and the second event has attribute *Name=Administratief* (Administration). On the other hand, cluster 1 is characterised by 3 events, the first one having attributes *Name=Vervolgconsult* (FollowupConsultation) and *ActivityCode=411100*. Further, cluster 0 is characterised by *AgeGroup=retired*, and cluster 1 by *AgeGroup=working*. However, from these subtrees one cannot ascertain whether any other events occurred between detected common events among the instances and/or whether the additional events differed. On the other hand, the frequent subtrees detected by *DSM-LCM* confirm that no additional events occurred between the common events of cluster 0, and that they were the first executed events within the traces (as inferred from node positions, see Fig. 4.3 and Fig. 4.4). In fact, the position-constrained subtrees also indicate how many events in total occurred in 90% of instances of the cluster, i.e. 3 events for cluster 0 (at locations X_4 , X_{12} and X_{20}) and 4 events for cluster 1 (at locations X_4 , X_{12} , X_{20} and X_{28}). Furthermore, the attribute *Name=Administratief* in the second event of the traditional subtree of cluster 0 in Fig. 4.1 did not occur in position-constrained subtree of cluster 0 in Fig. 4.3, which also indicates that this attribute was not frequent in the second event but was frequent when its occurrences in the second and third event were counted together. This has been confirmed by observing the instances of cluster 0 and have found that the attribute *Name=Administratief* occurs 7 times in the second event and 6 times in the third event. Another difference is that the attributes *Name=Vervolgconsult* and *ActivityCode=411100* did not occur in the position-constrained subtree of cluster 1. This can be explained by the fact that the association of *Name=Vervolgconsult* and *ActivityCode=411100* occurs 18 times in the first event and 7 times in the second event, which are less than the required minimum support of 25 ($= 90\% * 28$).

The observed differences indicate the benefit of the *PCFSM* method in two cases: (i) to find the exact location of each repeated or outlying value, and (ii) to know the exact occurrence of an event and its attributes within the trace as a whole. However, it is not claimed that traditional subtree mining would not be useful, but that each approach have its own useful characteristics. For example, the traditional frequent subtrees reflect the occurrences of events and their characteristics no matter in which part of the process instance they occurred, while the position-constrained frequent subtrees provide further detail of their exact locations within a trace. Both approaches could be used in a complementary way to obtain a more comprehensive analysis.

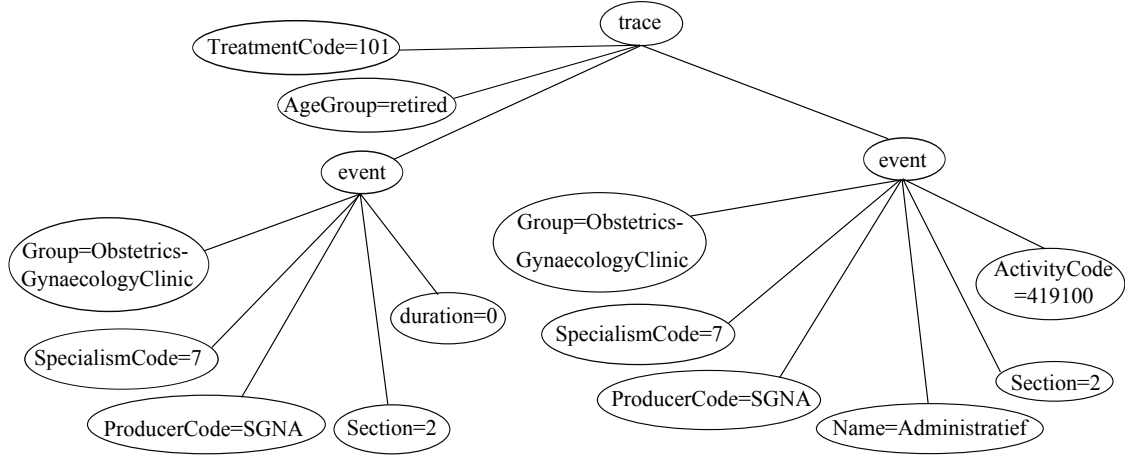


Figure 4.1: A frequent subtree in cluster 0.

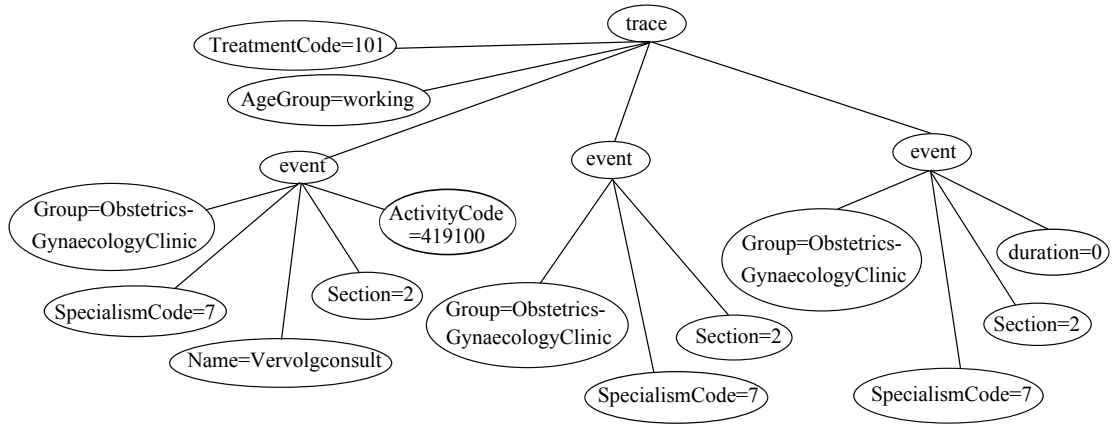


Figure 4.2: A frequent subtree in cluster 1.

4.2.3 Discovering Discriminating Characteristics of Groups

Classification methods can be used to identify key differences among group of process instances. The first three clusters are selected for the classification task since they have relatively high *ISim* and low *ESim* value. For each cluster, 70% of randomly selected instances are reserved for the training set and the remainder for the test set. Because the number of instances for each class is different, the oversampling method is used to balance the examples of each class. Random instances of cluster 0 and 1 are duplicated to reach the total number instances of cluster 2. In what follows, the decision tree learning and the associative classification methods are evaluated on these clusters.

Decision Tree Learning

The *C4.5* decision tree learning algorithm in the *RapidMiner* software (Mierswa et al., 2006) was used with default parameter settings (criterion: gain ratio, confidence: 0.5). Note that this setting is also used for all decision tree learning tasks in this chapter. The

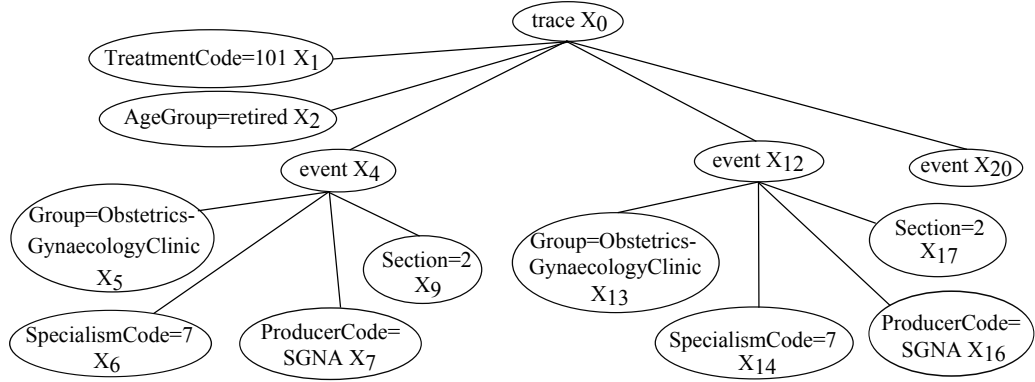


Figure 4.3: A position-constrained frequent subtree in cluster 0.

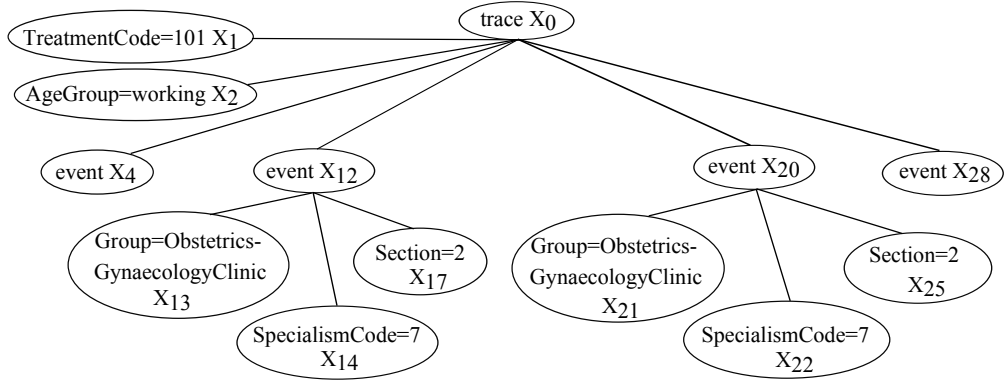


Figure 4.4: A position-constrained frequent subtree in cluster 1.

resulting decision tree is displayed in Fig. 4.5, which has an accuracy of 93.3%. The decision tree learning, model application, and performance evaluation were accomplished in under 1 second. An interesting rule found from the decision tree is that if $X_2 = \text{AgeGroup} = \text{retired}$, $X_{39} = \text{NO}$, then the instance is classified as belonging to cluster 0. Since the *PCFSM* method represents tree-structured data in a position-constrained format, it is always possible to identify the location of each data attribute according to the *DSM*. In the hospital dataset, each event has seven attributes (it was nine attributes before the data preparation phase). Hence, if the node *event* of the first event of a trace is located at X_4 , then the node *event* of the fourth event (if it exists) is located at X_{36} according to the *DSM*.

It is known that all attributes of an event must be present if that event exists. Hence, a value of *NO* in any position implies all *NO* values for all columns next to it. With this in mind, the above rule can be interpreted as “all patients who are in the retiring age group and there are no more than 4 events in their process should be put into the same group, which is cluster 0 in this case”.

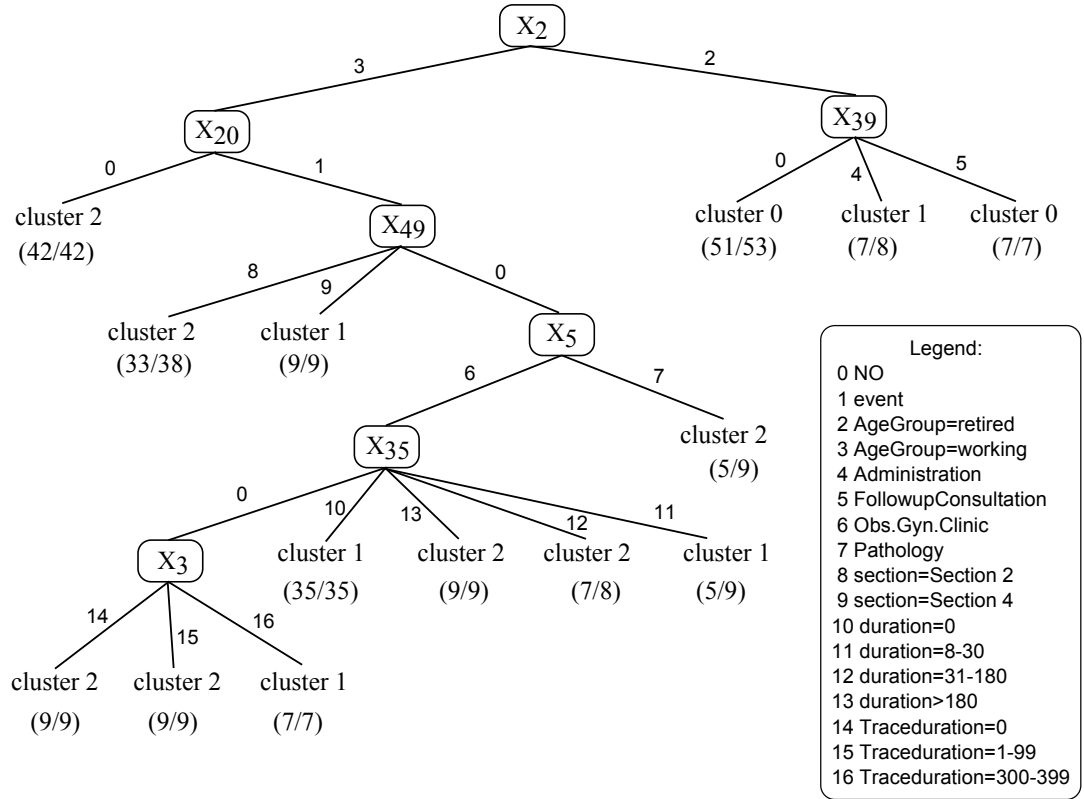


Figure 4.5: A decision tree learned from three clusters of process instances.

Associative Classification

In associative classification, the *FDT* data is first converted to itemset representation and frequent itemset mining algorithm is then used to find associations among the items. For classification purposes, only association rules that have the consequents as class labels are kept. For each rule, a rule strength (Zaki and Aggarwal, 2006) expressing the predicting power of that rule is computed. Note that in this experiment, the confidence measure was used to calculate the rule strength of each rule. Rules that do not satisfy the minimum threshold are removed. Finally, each instance in the dataset is evaluated against the set of rules and the class whose aggregated rule strength is the highest is selected as predicted value. More details of associative classification on *XML*-based data are provided in Chapter 5.

Table 4.4 shows the accuracy and coverage rates of each class. The best results were achieved at the lowest support threshold (2%) and the highest confidence value (90%). It is notable that the accuracy and coverage rates for cluster 0 are 100% for all settings. This can be explained by the high homogeneity of cluster 0 and its large similarity distance to other clusters ($ISim = 0.999$ and $ESim = 0.003$). Similarly, one can observe the gradual decrease in classification performance for clusters that have lower $ISim$ and higher $ESim$ values (cluster 1 and 2).

Table 4.4: Prediction accuracy and coverage rates for each class.

		cluster 0	cluster 1	cluster 2
support=2%	accuracy	100.0	89.5	58.8
confidence=50%	coverage	100.0	100.0	100.0
support=2%	accuracy	100.0	100.0	92.3
confidence=90%	coverage	100.0	89.5	76.5
support=10%	accuracy	100.0	89.5	35.3
confidence=50%	coverage	100.0	100.0	100.0
support=10%	accuracy	100.0	100.0	100.0
confidence=90%	coverage	100.0	78.9	17.6

4.3 Classification based on *PCFSM* on Labelled Dataset

In this section, the decision tree learning algorithm is applied to a subset of hospital dataset, which is described in (Bose and Aalst, 2012), and two synthetic datasets.

A Subset of the Hospital Dataset

In the first setting, process instances having diagnosis code *M13* and treatment code *803* are selected for our experiment (combinations with other treatment code, e.g. *803_101* are not considered). This filtered dataset has 23 process instances comprising two groups: the first group has 15 instances containing *non-urgent* cases, and the second group contains the remaining 8 *urgent* cases. The resulting decision tree is shown in Fig. 4.6. The classification accuracy (using ten-fold cross-validation) is 100% with all instances being covered. It can be seen that any case diagnosed with *maligniteit cervix* is identified as *urgent*.

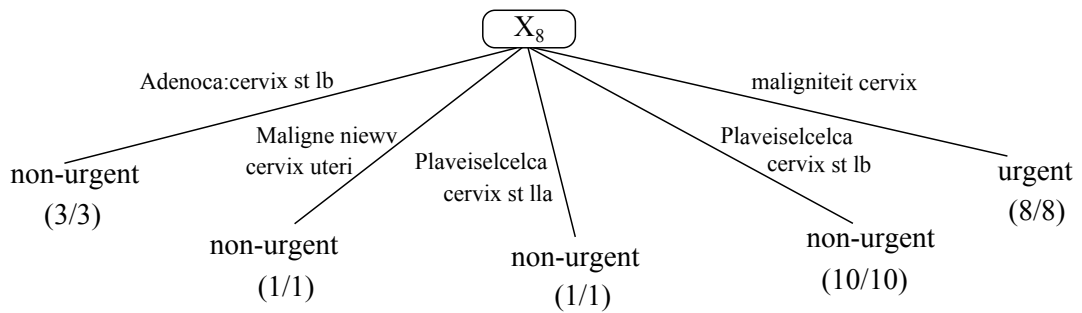


Figure 4.6: Urgency prediction of a process instance on diagnosis code *M13* and treatment code *803*.

In the second setting, the dataset contains only process instances having diagnosis code *M11*. There are 25 *urgent* and 137 *non-urgent* cases in this filtered dataset. When the decision tree learning algorithm is performed on this data and with ten-fold cross-validation, the classification accuracy achieved is at 92.59%. It can be seen from

Fig. 4.7 that at the position of X_{1152} which corresponds to the 139th event, if there is no further event, the case is classified as *non-urgent*. If there are more events after this event, then the first event attribute (at location X_{52}) is checked. If this event is administered in *Section 2* (e.g. where *ObstetricClinic* and *NursingWard* department reside), the case is classified as *urgent*; if it belongs to *Section 4* (e.g. where *Pathology* and *MedicalMicrobiology* reside), then it is classified as *non-urgent*. It can be seen that the exact points of difference between *urgent* and *non-urgent* cases are indicated. This kind of analysis is performed from a different perspective to the process discovery and trace alignment method (Bose and Aalst, 2012), and can be seen as complementary to both approaches.

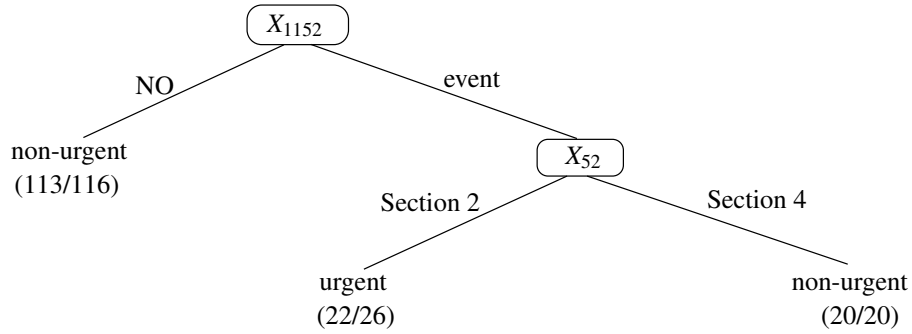


Figure 4.7: Urgency prediction of a process instance on diagnosis code *M11*.

Teleclaim Dataset

In the teleclaim dataset (van der Aalst, 2011b), the potential of the *PCFSM* method in classifying four possible outcomes of a claim such as *processed*, *rejected*, *insufficient information* or *not liable* is explored. Each trace is first labelled with one of the four values as described above. Note that the dataset is balanced using the oversampling method. The decision tree learning using ten-fold cross-validation returns an accuracy of 100% covering all cases. The decision tree in Fig. 4.8 shows that if activity *end* happens at the seventh event (at position X_{35}), the claim is classified as *not liable*; otherwise the claim would be *insufficient information*. Furthermore, at position X_{53} , it is known that if the eleventh event is available, the claim is categorised as *processed*, otherwise it is *rejected*. This indicates another useful property of the *FDT* representation, as specific points of difference between events of traces of different class can be directly detected.

Telephone Repair Dataset

In the telephone repair dataset (van der Aalst, 2011c), the objectives are to predict (i) the time needed to repair each telephone (class values are 0, 1 and 2 hours) and (ii) the complexity of the repair—which can be *simple*, *complex* or *both*. Note that the number

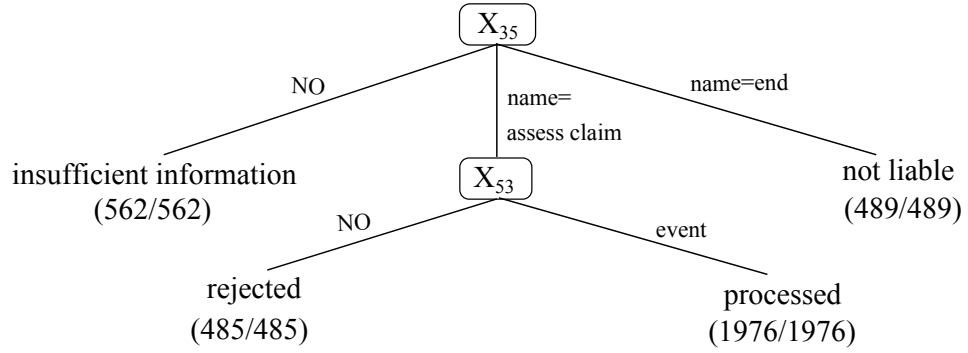


Figure 4.8: A decision tree for outcomes of an insurance claim in the *teleclaim* dataset.

of attributes of the events of this dataset is the same but some of them differ in order. Hence, the attributes were sorted in the same order to ensure that columns in the *FDT* format contain values of the same attribute.

Prediction of duration: The resulting decision tree model (took 6s to build and evaluate) has the size of 69 with 58 leaf nodes and an accuracy rate of 82.7%, which was evaluated by ten-fold cross-validation. Due to its large size, the decision tree is not shown here. Inspecting the rules of the decision tree shows that the duration of a process is classified as 0 (hour) if $X_{90} = X_{67} = X_{41} = \text{NO}$. Note that X_{90} contains the fourth attribute (*number of repair*) of the eighth event, X_{67} contains the the fifth attribute (*defect fixed*) of the thirteenth event and X_{41} contains the fifth attribute (*defect fixed*) of the eighth event. X_{27} , which corresponds to the first attribute, i.e *resource*) of the sixth event, names the officer responsible for the repair at that stage. From the decision model, it was observed that officer *SolverC3* performed poorly as most of his/her tasks are completed in one hour while others finish in less than one hour. Although only a synthetic dataset was used, this kind of analysis is also useful for detecting performance bottlenecks and investigating resource optimisation.

Prediction of complex process: a trace is labelled *simple* (or *complex*) if it contains any *simple* (or *complex*) repair, as part of the descriptive attributes of activities within an event. If both *simple* and *complex* repairs exist then the trace is labelled *both*. Part of the resulting decision tree for the three-class classification evaluated using ten-fold cross-validation is displayed in Fig. 4.9 and the classification accuracy achieved is at 92.94%.

4.4 Lessons Learned and General Remarks

The proposed approach, *PCFSM*, integrated existing algorithms/methods for the purpose of bringing about the first approach that allows the direct application of data mining techniques to tree-structured process logs, without the need of a process model. Although the persistence method of process log was standardised by *MXML/XES* and

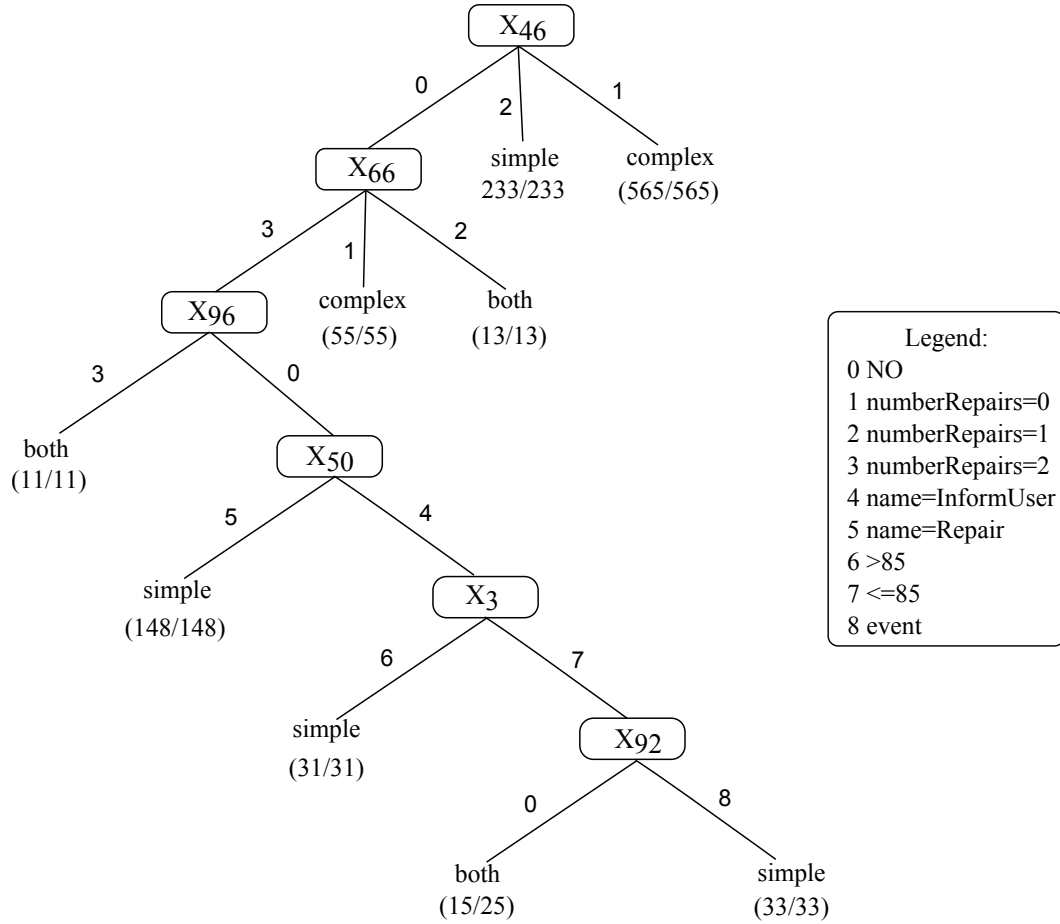


Figure 4.9: A decision tree for complexity prediction of the *telephone repair* dataset.

many *XML* mining techniques have been around for a long time, to our best knowledge, there has been no application of *XML* mining algorithms to event data.

The traditional tree-structured mining methods, such as *CMTreeMiner* and *Dryade-Parent*, which represent the state-of-the-art in the *FSM* field, typically have performance issues on process logs due to their large size and high complexity. On the other hand, the proposed method, which is based on the position-constrained frequent subtree mining, is able to partially overcome those obstacles by flattening out the tree structures, thereby simplifying the subtree mining task to that of itemset mining.

The experimental results confirm the three benefits of the *PCFSM* method, which were stated in Section 4.1. Firstly, the frequent subtree patterns discovered by the method showed the exact order and position of each event in the frequent path of executions. From this information, a process analyst can immediately pinpoint the events that occur frequently, instead of having to search for them if traditional methods are to be used. Secondly, the *PCFSM* method allowed for the applications of *CLUTO* (a clustering algorithm), *C4.5* (a decision tree learning algorithm) and *LCM* (a frequent itemset mining algorithm) to tree-structured process logs. Thirdly, no process models were involved in the process.

The advantage that by using *PCFSM*, the exact position of each event/node in the trace was preserved, comes with the potential cost in the data preparation process. An assumption of the method is that event attributes are organised uniformly across different process instances, as is often the case for *XML* documents conforming to a well-defined schema. However, since the *MXML* and *XES* standards do not ask for a strict ordering of event attributes, the conversion of tree-structured event logs whose attributes are not ordered, to a flat data format would lead to corresponding attributes being put into different columns of the table. This might severely affect the performance of data mining algorithms, which are used in the next steps. This problem can be avoided by standardising the structural organisation of the event attributes during pre-processing steps.

It was observed that the number of patterns discovered by the *PCFSM* method is often large, which leads to the difficulty in identifying interesting patterns. Moreover, trial and error had to be used to determine the best support threshold. However, these drawbacks are inherent to any method that is based on frequent pattern mining. Several techniques have been proposed in the data mining field to alleviate these problems such as pattern filtering based on interestingness measures.

It is also worth noting that due to the nature of our approach in exact matching of subtrees by node positions, some traditional frequent subtrees might be missed, depending on the selected support threshold. However, those subtrees can still be discovered by lowering the support threshold to a certain level, and then performing additional post-processing steps: (i) node positions of discovered patterns are removed and (ii) subtrees that are structurally-equivalent with all nodes sharing the same respective label are merged, and their supports are summed up. In any case, the *PCFSM* method presents a good alternative to traditional methods in complex data for which the frequent pattern mining task is infeasible.

An unsupervised, exploratory process log analysis method (*EPLA*) based on *PCFSM* was proposed. This method separates process instances into different groups having similar characteristics. The experimental results indicated that similar properties such as process execution time, cost, and execution order are shared among instances of each group.

The clustering task enabled *FSM* techniques, such as *CMTreeMiner*, and *DSM-LCM* to be applied to each group of instances, in order to explore their descriptive characteristics. Furthermore, it is possible to investigate the distinguishing characteristics of each group by applying decision tree learning or associative classification algorithm to instances labelled respective to the groups being analysed. In practice, it is typical to have a conclusion like “this group of process instances is different from other groups in that most patients are retired and the majority of the treatment processes last at least ten months”. In general, the *EPLA* method allows us to analyse the

process log in an unbiased way when there is a lack of domain knowledge to guide the analysis process.

The evaluation of the *PCFSM* method on supervised setting showed that this approach has the potential in uncovering interesting and actionable knowledge from the process log. For example, based on the analysis of a labelled event log of a hospital, the proposed method might suggest a business rule that says “any treatment process that has more than 139 events in less than one day is considered an urgent case and should be notified to relevant doctors”. In the synthetic telephone repair dataset, by labelling each process instance with its duration, a decision tree learning algorithm might be able to uncover workers whose performances are poor. Our approach adds to the available pool of performance analysis methods, such as the dotted chart method (Song and van der Aalst, 2007).

Chapter 5

A Position-constrained Associative Classification Approach

This chapter presents an extension of the *PCFSM* method for the classification problem. The structure of this chapter is outlined as follows. Section 5.1 proposes an associative classifier, called *DSMC*, which is based on the positioned-constrained frequent/closed subtree mining. The experimental settings and data characteristics are described in Section 5.2. In Section 5.3, *DSMC* is applied to heterogeneous real-world and synthetic tree-structured datasets whose structures are varied among instances. The evaluation continues in Section 5.4 with homogeneous data, where a real-world process log and a real estate *XML*-based dataset are examined. Section 5.5 and ?? describe the use of *DSMC* to predict outcomes of running process instances or to recommend actions that have higher chance of leading to a desired outcome. Enhancements for the *DSMC* classifier are discussed in Section 5.7. Finally, Section 5.8 concludes this chapter.

5.1 Associative Classification on Process Logs

By representing tree-structured data in a flat representation (as part of the *PCFSM* method), traditional algorithms such as *CART* or *C4.5* can be utilised for the classification task on tree databases. This data re-representation also enables the associative classification framework (Liu et al., 1998) to be adapted for the tree-structured data classification. To do this, the flat representation is first transformed to itemset format, which is then fed to a frequent itemset mining algorithm. From the resulting frequent itemsets, one could form association rules with consequents as class labels. The antecedents can be remapped back into subtrees, so classification rules are now in the form of subtrees. It has been shown that classification rules are comprehensible to users and in general rule-based models tend to give better accuracy and are more

scalable than traditional methods, e.g. *C4.5* (Han and Kamber, 2006).

In this section, two approaches for event data classification are discussed. First, an associative classification method which is based on traditional frequent subtree mining (Zaki and Aggarwal, 2006) is introduced. Second, an associative classifier which is based on position-constrained frequent subtree mining (Hadzic, 2012) is proposed. This classification method is developed as an extension to the *PCFSM* method.

5.1.1 Frequent Subtree-based Classification

In frequent subtree-based classification, the purpose is to discover from the training data rules (R) that have the form of $st \Rightarrow C$ where st is a subtree and C is the class label. The set of all rules is used to classify unseen instances.

Rules that have low discriminating power should be removed from the rule set in order to improve the classification accuracy, running performance, and comprehensibility of the classification model. This can be achieved by pruning rules that do not pass a user-specified minimum rule support (defined in Section 2.3.9) threshold and/or rule interestingness (e.g. confidence) threshold.

In addition to rule support, *class support* is another measure used to prune low discriminating power rules when the classes of the dataset are unbalanced. The class support was used in the work of Zaki and Aggarwal (2006).

Definition 5.1 (Class Support). Let $\pi^A(st, C)$ be the number of instances that contains subtree st which is labelled as C . A class support of a rule $st \Rightarrow C$ is defined as the ratio between $\pi^A(st, C)$ and the number of instances of class C .

$$\mathfrak{s}(st \Rightarrow C) = \frac{\pi^A(st, C)}{|C|}.$$

Definition 5.2 (Confidence). Let $\pi^A(st)$ be absolute support of subtree st in a database. The confidence of a rule ($st \Rightarrow C$) is defined as a ratio between $\pi^A(st, C)$ and $\pi^A(st)$.

$$\mathfrak{c}(st \Rightarrow C) = P(C|st) = \frac{\pi^A(st, C)}{\pi^A(st)}.$$

Definition 5.3 (Weighted Confidence). Let $\pi(st, C)$ be the support of subtree st in class C and $\pi(st, \bar{C})$ be the support of st in other classes). The weighted confidence of a rule ($st \Rightarrow C$) is defined as follows

$$\mathfrak{wc}(st \Rightarrow C) = \frac{\pi(st, C)}{\pi(st, C) + \pi(st, \bar{C})}.$$

Definition 5.4 (Likelihood). *The likelihood of a rule ($st \Rightarrow C$) is defined as follows*

$$l(st \Rightarrow C) = \frac{\pi(st, C)}{\pi(st, \bar{C})}.$$

Rule strength is a measure that represents the discriminating power of a classification rule. The rule strength of rule $st \Rightarrow C$ is defined as follows.

Definition 5.5 (Rule strength). $\rho_\eta(st \Rightarrow C) = \eta(st \Rightarrow C)$,

where η is any type of interestingness measure (see 2.3.14) that quantifies the relation between subtree st and class C : $\eta = \{c, wc, l, \dots\}$.

For example, the confidence-based rule strength is defined as $\rho(st \Rightarrow C) = c(st \Rightarrow C)$. The weighted-confidence- and likelihood-based rule strength are defined as $\rho(st \Rightarrow C) = wc(st \Rightarrow C)$ and $\rho(st \Rightarrow C) = l(st \Rightarrow C)$, respectively.

To predict unseen data, each instance is matched against each rule in the rule set. If the antecedent of a rule occurs in the test instance, it can be said that the rule is triggered by that instance. Let t be an unseen instance, $\{R_1 \dots R_m\}$ be the rules that are triggered by t , p be the number of class values, and $R_i : st_i \Rightarrow C_k, i \in \{1 \dots m\}, k \in \{1 \dots p\}$ be one of the rules, the combined rule strength on instance t for class $C_x, x \in \{1 \dots p\}$ is defined as

Definition 5.6 (Combined rule strength). $\rho_{\eta, C_x}(t) = \frac{\sum_{i=1}^m \rho_\eta(R_i)}{m} = \frac{\sum_{i=1}^m \rho_\eta(st_i \Rightarrow C_x)}{m}$.

A test instance t is classified as class C_x if $\rho_{\eta, C_x}(t) > \rho_{\eta, C_y}(t), \forall y \neq x$, with $x, y \in \{1 \dots p\}$.

To evaluate the predictive capability of the rule set on a test data, classification results are compared against actual labels of the test data. Evaluation criteria include accuracy and coverage rates, and the overall accuracy and coverage rates.

Let n_{C_i} be the number of instances that belong to C_i and are covered by the rule set. Let m_{C_i} be the number of instances that belong to C_i and are correctly classified. For illustration, Fig. 5.1 shows a database with two class labels C_1 and C_2 . This figure expresses three aspects of a classification evaluation: (i) the number of instances for each class, (ii) the number of instances that are covered by a rule set, and (iii) the number of instances that are correctly classified.

Definition 5.7 (Class coverage). *The coverage rate of class C_i is calculated as follows*

$$CR(C_i) = \frac{n_{C_i}}{|C_i|}.$$

Definition 5.8 (Class accuracy). *The accuracy rate of class C_i is calculated as follows*

$$AR(C_i) = \frac{m_{C_i}}{n_{C_i}}.$$

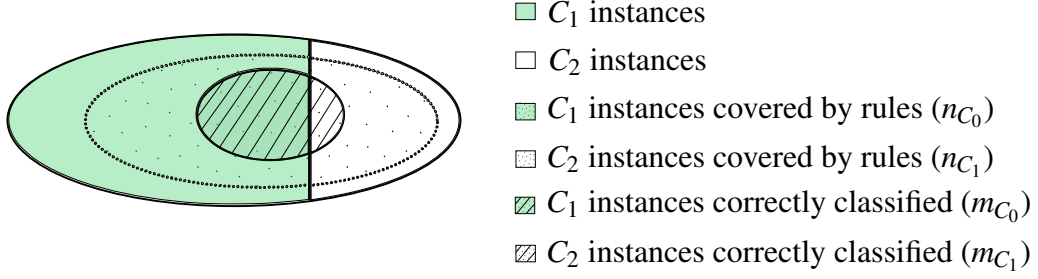


Figure 5.1: An illustration for classification evaluation.

Definition 5.9 (Overall coverage). *The overall coverage of a rule set is defined as follows*

$$CR = \frac{\sum_{i=1}^p n_{C_i}}{\sum_{i=1}^p |C_i|}.$$

Definition 5.10 (Overall accuracy). *The overall accuracy of a rule set is defined as follows*

$$AR = \frac{\sum_{i=1}^p m_{C_i}}{\sum_{i=1}^p n_{C_i}}.$$

5.1.2 DSMC—the Proposed Classification Method

In this section, *DSMC*, an associative classifier which is based on position-constrained frequent subtree mining (Bui et al., 2013, 2014) is proposed. This classifier expands phase IV of the *PCFSM* method. Essentially, *DSMC* is a six-step procedure which is described as follows.

1. Convert to itemset format: if the tree-structured database is in the *FDT* format, the data items of each row are attached to their corresponding columns and the class label is added to the itemset representation of each tree;
2. Discover frequent itemsets for each class: in many cases the classes in the training dataset are unbalanced in terms of the number of instances, which might lead to rules related to the minor class being under-represented in the rule set. For this reason, the dataset is separated into partitions, each of which associates to a class label. Next, a frequent subtree mining algorithm is applied to each partition;
3. Form class-associative rules: each itemset discovered in the previous step represents an association rule with antecedent being the frequent itemsets and the rule consequent being the class label. The set of all discovered class-associative rules forms the classification model;
4. Identify the rule strength for each rule in the rule set;

5. Find rules that are triggered by each test instance: to determine which rule can be triggered by an instance, the position-constrained frequent subtree in the antecedent part of each rule is matched against the flat representation of the test instance. The conversion of tree structure to tabular format references the *DSM* of the training data instead of the testing data to ensure that the positions of tree nodes are labelled consistently;
6. Calculate the combined rule strength per class for each instance: the predicted class value is the one that has highest combined rule strength.

5.1.3 Illustration of the Proposed Classification Method

In this section, a concrete example is given to illustrate the two classification approaches discussed in previous sections.

Let T_{cl} be a tree-structured database shown in Fig. 5.2. T_{cl} includes four tree instances t_1, t_2, t_3, t_4 with respective class labels C_1, C_1, C_2, C_2 .

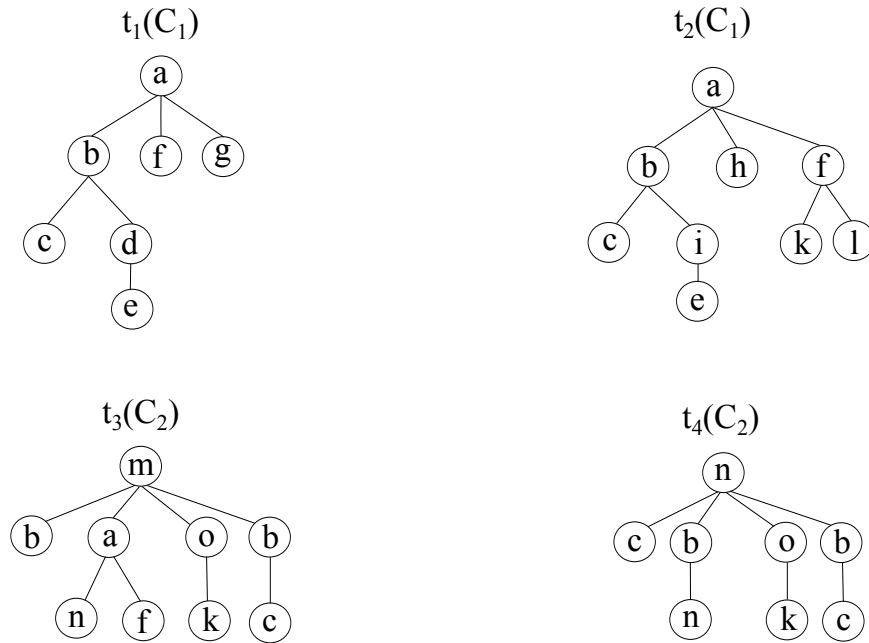


Figure 5.2: Tree database T_{cl} .

In both traditional and position-constrained frequent-subtree-mining-based classification methods, it is assumed that the class support threshold is 50% and the confidence threshold is also 50%.

Associative Classifier based on Frequent Subtree Mining

A subset of all frequent subtrees of T_{cl} that satisfy the class support threshold is shown in Fig. 5.3. Note that subtrees st_1, st_3 and st_4 are also induced subtrees, as all nodes are

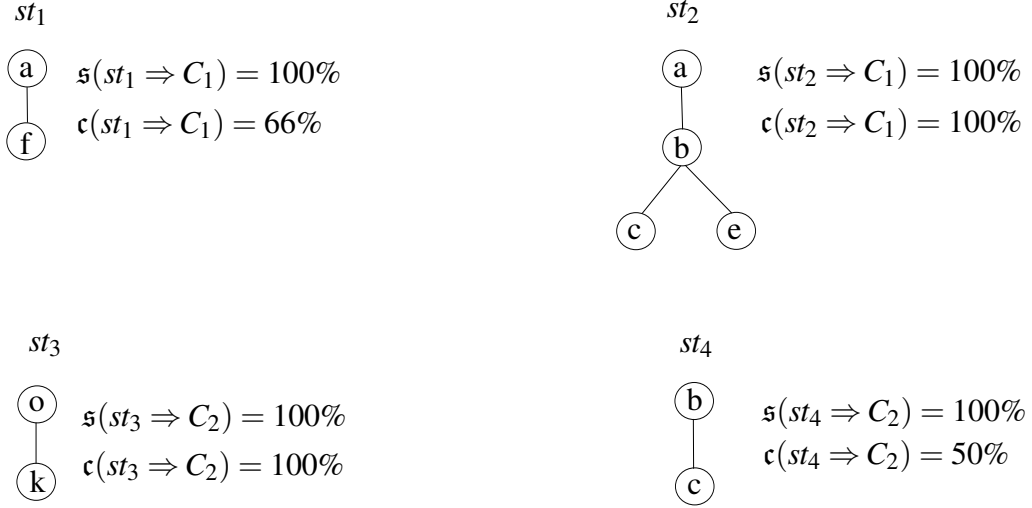


Figure 5.3: Four frequent embedded subtrees of T_{cl} and their corresponding supports.

separated by a single edge in the original trees where they occur (i.e. st_1 is supported by t_1 , t_2 and t_3 ; st_3 by t_3 and t_4 ; st_4 by all trees). This is not the case for subtree st_2 because in t_1 and t_2 where it occurs there are two edges separating nodes b and e , (i.e. there is an embedded relationship between b and e).

Class-associative rules are obtained from the frequent subtrees. Supposedly, only four rules are selected for the rule set. These rules and their corresponding confidence-based rule strength are listed as follows.

- $R_1 : st_1 \Rightarrow C_1$
 $\rho_c(R_1) = c(R_1) = \frac{2}{3} = 66\%$
- $R_2 : st_2 \Rightarrow C_1$
 $\rho_c(R_2) = c(R_2) = \frac{2}{2} = 100\%$
- $R_3 : st_3 \Rightarrow C_2$
 $\rho_c(R_3) = c(R_3) = \frac{2}{2} = 100\%$
- $R_4 : st_4 \Rightarrow C_2$
 $\rho_c(R_4) = c(R_4) = \frac{2}{4} = 50\%$

Let T_{eval} (shown in Fig. 5.4) be a test database used for classification evaluation. The transactions in T_{eval} are evaluated using the confidence-based rule strength. The detail of the evaluation is presented as follows.

- $\rho_{c,C_1}(t_a) = \frac{\rho_c(st_1 \Rightarrow C_1) + \rho_c(st_2 \Rightarrow C_1)}{2} = \frac{66\% + 100\%}{2} = 83\%.$
 $\rho_{c,C_2}(t_a) = \frac{\rho_c(st_3 \Rightarrow C_2) + \rho_c(st_4 \Rightarrow C_2)}{2} = \frac{100\% + 50\%}{2} = 75\%.$
 Since $\rho_{c,C_2}(t_a) < \rho_{c,C_1}(t_a)$, t_a is classified as C_1 .

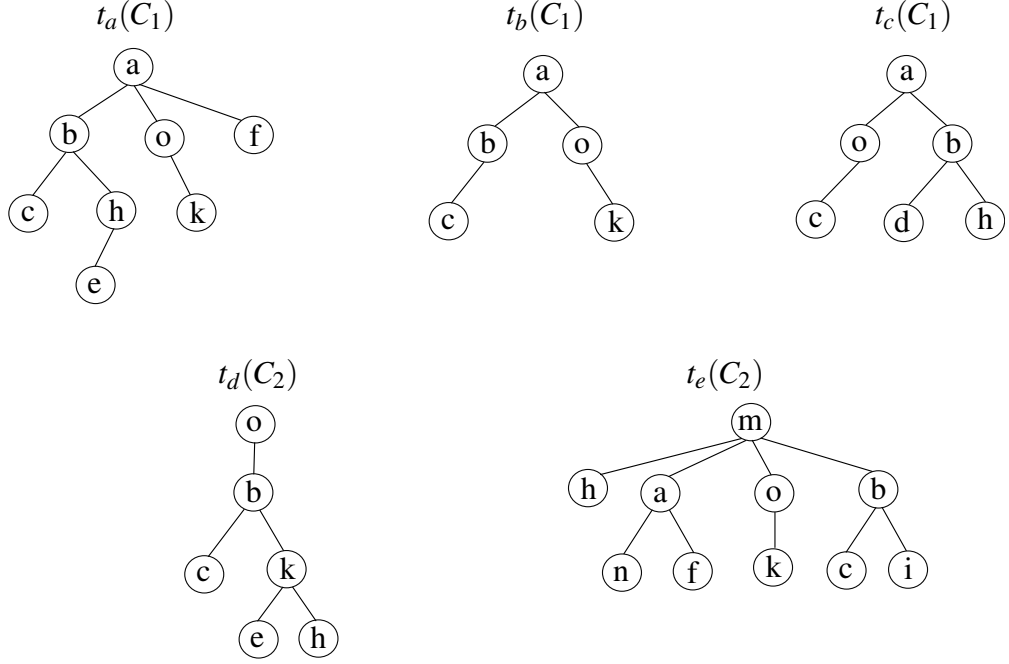


Figure 5.4: T_{eval} —a synthetic tree database used for classification evaluation.

- $\rho_{c,C_1}(t_b) = 0\%$

$$\rho_{c,C_2}(t_b) = \frac{\rho_c(st_3 \Rightarrow C_2) + \rho_c(st_4 \Rightarrow C_2)}{2} = \frac{100\% + 50\%}{2} = 75\%$$

Since $\rho_{c,C_2}(t_b) > \rho_{c,C_1}(t_b)$, t_b is classified as C_2 .

- $\rho_{c,C_1}(t_c) = \rho_{c,C_2}(t_c) = 0$ since there are no subtrees in the frequent embedded subtree set st_1, st_2, st_3, st_4 occur in t_c .

- $\rho_{c,C_1}(t_d) = 0\%$

$$\rho_{c,C_2}(t_d) = \frac{\rho_c(st_3 \Rightarrow C_2) + \rho_c(st_4 \Rightarrow C_2)}{2} = \frac{100\% + 50\%}{2} = 75\%$$

Since $\rho_{c,C_2}(t_d) > \rho_{c,C_1}(t_d)$, t_d is classified as C_2 .

- $\rho_{c,C_1}(t_e) = \rho_c(st_1 \Rightarrow C_1) = 66\%$

$$\rho_{c,C_2}(t_e) = \frac{\rho_c(st_3 \Rightarrow C_2) + \rho_c(st_4 \Rightarrow C_2)}{2} = \frac{100\% + 50\%}{2} = 75\%$$

Since $\rho_{c,C_2}(t_e) > \rho_{c,C_1}(t_e)$, t_e is classified as C_2 .

The evaluation results of the test data shown in Fig. 5.4 are presented as follows.

- $CR(C_1) = \frac{2}{3} = 66\%$, $AR(C_1) = \frac{1}{2} = 50\%$

- $CR(C_2) = \frac{2}{2} = 100\%$, $AR(C_2) = \frac{2}{2} = 100\%$

- $CR = \frac{4}{5} = 80\%$, $AR = \frac{3}{4} = 75\%$

Associative Classifier based on Position-constrained Frequent/Closed Subtree Mining

The *DSM* of the tree database T_{cl} is presented in Fig. 5.5. For illustration, the *DSM* used for the classification task is extracted from the database at the minimum support of 50%. This *DSM* is shown in Fig. 5.6. Note that two nodes at locations X_7 and X_{10} of the original *DSM* are safely removed due to their low frequency of occurrence (according to the support threshold).

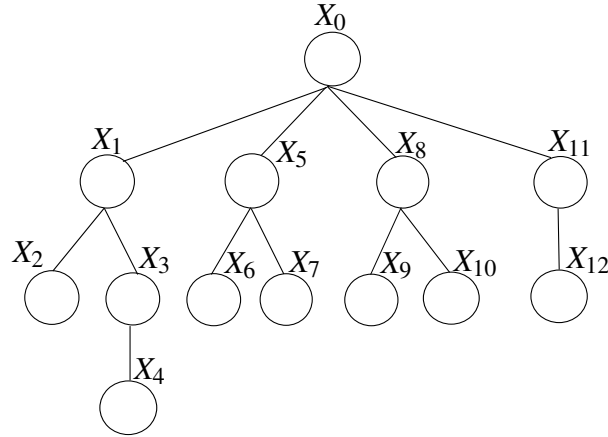


Figure 5.5: The *DSM* tree of T_{cl} .

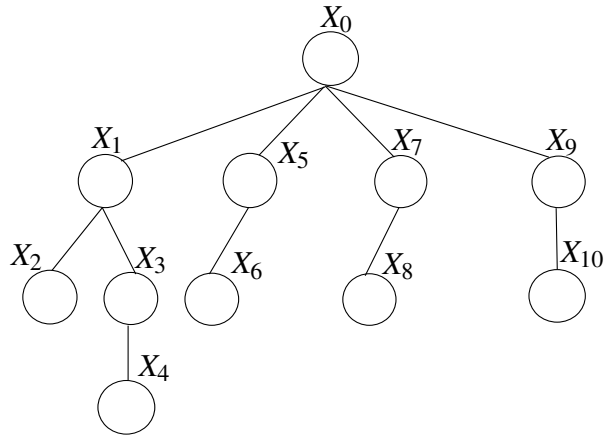


Figure 5.6: The *DSM* tree at minimum support = 50% of T_{cl} .

In the next step, the tree-structured database is converted to a tabular format. The *FDT* representation of T_{cl} is shown in Table 5.1. Its corresponding itemset representation is shown in Table 5.2. Note that the data instances are split into two partitions according to their class labels.

Then, a frequent itemset mining algorithm is applied to each partition of Table 5.2. The minimum support for the frequent itemsets should be similar to the class support of frequent subtree, which is 50%. Two frequent itemsets that meets the minimum support requirements for partition C_1 are $\{bX_1, cX_2\}$ and $\{bX_1, cX_2, eX_4\}$. Two frequent itemset

Table 5.1: The flat data representation of T_{cl} .

t_i	X_0	X_1	X_2	b_0	X_3	X_4	b_1	b_2	b_3	X_5	X_6	b_4	b_5	X_7	X_8	b_6	b_7	X_9	X_{10}	b_8	b_9
t_1	a	b	c	1	d	e	1	1	1	f	0	0	1	g	0	0	1	0	0	0	0
t_2	a	b	c	1	i	e	1	1	1	h	0	0	1	f	k	1	1	0	0	0	0
t_3	m	b	0	0	0	0	0	0	1	a	n	1	1	o	k	1	1	b	c	1	1
t_4	n	c	0	0	0	0	0	0	1	b	n	1	1	o	k	1	1	b	c	1	1

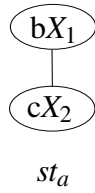
 Table 5.2: The itemset format of T_{cl} .

T_{cl}	Itemsets	Partition
t_1	$aX_0, bX_1, cX_2, dX_3, eX_4, fX_5, gX_7$	C_1
t_2	$aX_0, bX_1, cX_2, iX_3, eX_4, hX_5, fX_7, kX_8$	
t_3	$mX_0, bX_1, aX_5, nX_6, oX_7, kX_8, bX_9, cX_{10}$	C_2
t_4	$nX_0, cX_1, bX_5, nX_6, oX_7, kX_8, bX_9, cX_{10}$	

that meets the minimum support requirements for partition C_2 are $\{bX_9, cX_{10}\}$ and $\{nX_6, oX_7, kX_8, bX_9, cX_{10}\}$. Fig. 5.7 shows the reconstructed trees of the four frequent itemsets and their corresponding supports. Note that the frequent itemset $\{nX_6, oX_7, kX_8, bX_9, cX_{10}\}$ corresponds to a disconnected subtree.

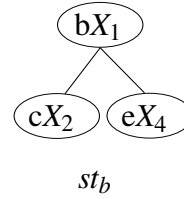
$$\rho_{c,C_1}(st_a) = 100\%$$

$$\mathfrak{s}(st_a \Rightarrow C_1) = 100\%$$



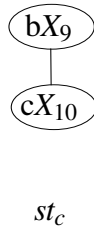
$$\rho_{c,C_1}(st_b) = 100\%$$

$$\mathfrak{s}(st_b \Rightarrow C_1) = 100\%$$



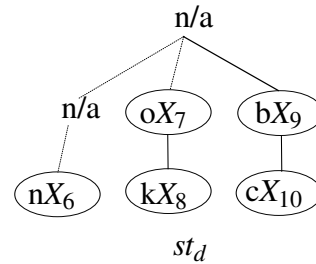
$$\rho_{c,C_2}(st_c) = 100\%$$

$$\mathfrak{s}(st_c \Rightarrow C_2) = 100\%$$



$$\rho_{c,C_2}(st_d) = 100\%$$

$$\mathfrak{s}(st_d \Rightarrow C_2) = 50\%$$


 Figure 5.7: Four position-constrained embedded subtrees of T_{cl} and their corresponding supports and confidences.

The class-associative rules are obtained by assigning the frequent itemsets as antecedents and their corresponding class labels as rule consequents. The rules and their confidence-based rule strengths are listed as follows.

- $r_1: st_a \Rightarrow C_1$
 $\rho_c(r_1) = \mathfrak{c}(r_1) = 100\%$

Table 5.3: The FDT representation of T_{eval} .

t_i	X_0	X_1	X_2	b_0	X_3	X_4	b_1	b_2	b_3	X_5	X_6	b_4	b_5	X_7	X_8	b_6	b_7	X_9	X_{10}	b_8	b_9
t_a	a	b	c	1	h	e	1	1	1	o	k	1	1	f	0	0	1	0	0	0	0
t_b	a	b	c	1	0	0	0	0	1	o	k	1	1	0	0	0	0	0	0	0	0
t_c	a	o	c	1	0	0	0	0	1	b	d	1	1	0	0	0	0	0	0	0	0
t_d	o	b	c	1	k	e	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0
t_e	m	h	0	0	0	0	0	0	1	a	n	1	1	o	k	1	1	b	c	1	1

- $r_2: st_b \Rightarrow C_1$
 $\rho_c(r_2) = \mathfrak{c}(r_2) = 100\%$
- $r_3: st_c \Rightarrow C_2$
 $\rho_c(r_3) = \mathfrak{c}(r_3) = 100\%$
- $r_4: st_d \Rightarrow C_2$
 $\rho_c(r_4) = \mathfrak{c}(r_4) = 100\%$

The next step is to represent the test data in itemset format using the DSM of the training data. The converted data in tabular and itemset format are shown in Table 5.3 and Table 5.4, respectively.

 Table 5.4: The itemset format of T_{eval} .

T_{cl}	Itemsets
t_a	$aX_0, bX_1, cX_2, hX_3, eX_4, oX_5, kX_6, fX_7$
t_b	$aX_0, bX_1, cX_2, oX_5, kX_6$
t_c	$aX_0, oX_1, cX_2, bX_5, dX_6$
t_d	$oX_0, bX_1, cX_2, kX_3, eX_4, kX_8, bX_9, cX_{10}$
t_e	$mX_0, hX_1, aX_5, nX_6, oX_7, kX_8, bX_9, cX_{10}$

The transactions of T_{eval} are evaluated using the confidence-based rule strength. The details of the evaluation are presented as follows.

- $\rho_{c,C_1}(t_a) = \frac{\rho_c(st_a \Rightarrow C_1) + \rho_c(st_b \Rightarrow C_1)}{2} = \frac{100\% + 100\%}{2} = 100\%$.
 $\rho_{c,C_2}(t_a) = 0\%$.
 Since $\rho_{c,C_2}(t_a) < \rho_{c,C_1}(t_a)$, t_a is classified as C_1 .
- $\rho_{c,C_1}(t_b) = \rho_c(st_a \Rightarrow C_1) = 100\%$
 $\rho_{c,C_2}(t_b) = 0\%$
 Since $\rho_{c,C_2}(t_b) < \rho_{c,C_1}(t_b)$, t_b is classified as C_1 .
- $\rho_{c,C_1}(t_c) = \rho_{c,C_2}(t_c) = 0$ since st_a, st_b, st_c and st_d do not occur in t_c .
- $\rho_{c,C_1}(t_d) = \frac{\rho_c(st_a \Rightarrow C_1) + \rho_c(st_b \Rightarrow C_1)}{2} = \frac{100\% + 100\%}{2} = 100\%$
 $\rho_{c,C_2}(t_d) = 0\%$
 Since $\rho_{c,C_2}(t_d) < \rho_{c,C_1}(t_d)$, t_d is classified as C_1 .

- $\rho_{c,C_1}(t_e) = 0\%$
 - $\rho_{c,C_2}(t_e) = \frac{\rho_c(st_c \Rightarrow C_2) + \rho_c(st_d \Rightarrow C_2)}{2} = \frac{100\% + 100\%}{2} = 100\%$
- Since $\rho_{c,C_2}(t_e) > \rho_{c,C_1}(t_e)$, t_e is classified as C_2 .

The evaluation results when applying *DSMC* to the synthetic test data are presented as follows.

- $CR(C_1) = \frac{2}{3} = 66\%, AR(C_1) = \frac{2}{2} = 100\%$
- $CR(C_2) = \frac{2}{2} = 100\%, AR(C_2) = \frac{1}{2} = 50\%$
- $CR = \frac{4}{5} = 80\%, AR = \frac{3}{4} = 75\%$

5.2 Experimental Settings

The main objective of the evaluation is to compare the associative classifier which is based on position-constrained frequent subtree mining (*DSMC*) with the state-of-the-art associative classifier which is based traditional frequent subtree mining (*XRules*).

The methods are evaluated using different minimum support values. The lowest support threshold to be selected should not be too small that it takes unreasonable time (more than a day) for the program to terminate or the memory to be exhausted. On the other hand, the highest support threshold to be selected should not be too high that the rule set becomes empty. Note that class support (see Definition 5.1) is used instead of rule support (see Definition 2.4) because most datasets are class-imbalanced and setting a low minimum support would result in the minority class being under-represented in the rule set.

Similar to the *XRules* evaluation presented in (Zaki and Aggarwal, 2006), the default rule strength thresholds, i.e. $(\mathfrak{w})c = 50\%$ or $\mathfrak{l} = 1$ are used in our experiment. For the sake of studying the effect of increasing the (weighted) confidence or likelihood threshold on the overall accuracy and coverage rate, the thresholds $(\mathfrak{w})c = 90\%$ or $\mathfrak{l} = 10$ are also used.

The frequent pattern types including frequent and closed are abbreviated as *Frq* and *Clo*, respectively. The subtree types including embedded and induced subtrees are abbreviated as *Emb* and *Ind*, respectively. The *DSMC* approach allows for another subtree option which is embedded-plus-disconnected subtree (abbreviated as *Eds*). Each run of the experiment is represented by a concatenation of different parameters, e.g. *DSMC.Frq.Emb.c* corresponds to *DSMC* using frequent embedded subtree setting and a confidence-based rule strength.

In cases *XRules* has trouble accomplishing the task due to node repetitions, i.e. nodes that share similar labels, all sibling nodes are assigned unique labels. The modified dataset is named by concatenating the original name with *SR*. To search for closed subtrees, the *DSMC* approach uses closed itemset mining, whereas the *XRules* implementation is adjusted so that *CMTreeMiner* is used instead of its default *Xminer* algorithm (Zaki and Aggarwal, 2006). The *DSMC* method uses the *FPGrowth* algorithm to generate (closed) frequent itemsets from the flat representation, whereas *CMTreeMiner* directly mines ordered closed induced subtrees. There are certain options that are not compatible with *XRules* or *DSMC*, for example, *Eds* cannot be used with *XRules* and the *SR* data should not be evaluated using the *DSMC* method because each node in the flat representation is already unique (the nodes are attached with positional information).

Data Characteristics

The real-world datasets used in this chapter are the hospital, *CSLOG*, and *CRM* datasets. The original hospital dataset was described in Section 4.1. To evaluate our proposed classification method and *XRules*, only a subset of the hospital dataset where processes with a single treatment code, i.e. '101' and single diagnosis codes, i.e. 'M13' and 'M16' are examined. These diagnosis codes describing two different types of cancer are selected as class labels. The classification rules obtained for each diagnosis code could uncover subtle differences in the treatment process of two rather similar types of cancer. Hence, the method is potentially useful for resource optimisation, e.g. assigning relevant resources for each diagnosis code.

The *CSLOG* dataset contains Web access trees of an academic institute during three weeks (Zaki and Aggarwal, 2006). The weekly subsets are as *CSLOG1*, *CSLOG2* and *CSLOG3* for each week. It contains user sessions labelled as *edu* or *other* depending on from where the web site was accessed from. For the purpose of classification, one *CSLOG* dataset is used as the training data and one other as the testing data; *CSx-y* denotes that *CSLOGx* is the training set and *CSLOGy* is the test set. Note that *edu* is the minority class in all three *CSLOG* datasets and the root nodes of all instances are removed as suggested in (Zaki and Aggarwal, 2006).

The *CRM* dataset is real-life data containing property management records from a real estate company¹. The data are originally stored in an *XML* format, which contain multiple cases, and each of which represents a maintenance record. Each case includes a set of elements such as 'Property' (property that needs maintenance), 'Contractor' (who actually does the maintenance) and 'Defect Data' (containing the description of all defects requested for maintenance). Each instance is labelled according to the in-

¹Due to confidentiality, the source of this dataset cannot be disclosed

formation derived from the tag ‘*TicketDuration*’. One class has duration of less than a month and the other has duration of more than a month. In practice, by investigating the classification rules, one can possibly uncover different hidden causes of delay, e.g. certain contractors or types of defects. This dataset contains many repetitive nodes called ‘*defect*’, which might compromise the performance of the frequent subtree mining task.

For synthetic datasets, called *sdb(s)*, a pseudo-random tree generator was developed using the following steps.

1. A number of random trees are generated and designated as subtree patterns.
2. A set of *coordinates* which represent the positions of nodes in a tree (e.g. t) are generated. The coordinates are used to locate the nodes where subtree patterns (e.g. p) can be attached to. Each coordinate includes a set of integers representing the sibling order of the nodes that run from the root node of t to p . For example, a node (n) whose coordinate in t is $[3,2]$ has $\alpha(n) = 2$, $\alpha(m) = 3$ where m is the parent of n , and m is the child of the root node of t .
3. A set of trees (sdb_1) is labelled as C_1 . Each tree in sdb_1 are randomly generated such that it contains nodes whose coordinates are a subset of the coordinates obtained in the previous step.
4. For each tree in sdb_1 , a random number of subtree patterns created in the first step are attached to random nodes whose coordinates are specified in the second step.
5. For each tree in sdb_1 , random subtrees are attached to nodes whose coordinates are not specified in the second step.
6. A set of trees (sdb_2) are labelled as C_2 and are created in the same manner as sdb_1 , except that a different set of coordinates is used.

The main characteristics of the tree generator described above are that node labels, trees’ height and fan-out, edges and subtree patterns are created randomly, but the subtree patterns are positioned among a set of random locations. Three synthetic datasets, i.e. *ST1*, *ST2*, and *ST3*, were created for experiments. The first one has 1.000 instances and each of the other two has 10.000 instances. Each of the first two datasets has the maximum of 100 distinct labels, whereas the remaining dataset has 300 distinct labels. The structural characteristics of the datasets used in this chapter are shown in Table 5.5. Note that *Maj. Per.* represents the percentage of the majority class in the whole dataset.

Table 5.5: Structural characteristics of the datasets used in Chapter 5.

Dataset	Tr	Avg L	Avg D	Avg F	Avg T	Max D	Max F	Max T	Maj.Per.
CS1-2	8074	625	3.4	1.9	8.0	123	130	313	75.7
CS2-3	7409	341	3.5	1.9	8.1	171	137	171	77.2
CS3-1	7628	383	3.4	1.9	8.0	120	130	192	76.4
Hospital Training	160	232	2.0	7.2	116.6	2	126	988	58.1
Hospital Test	79	210	2.0	7.2	105.4	2	62	476	62.8
CRM Training	900	95	4.4	7.1	47.8	5	32	320	66.4
CRM Test	281	91	4.3	7.2	45.9	5	34	338	66.6
ST1 Training	800	110	5.2	2.4	55.5	6	6	125	50.0
ST1 Test	200	110	5.2	2.4	55.6	6	6	116	50.0
ST2 Training	8000	123	5.9	2.4	62.2	6	6	132	50.0
ST2 Test	2000	123	5.9	2.4	62.4	6	6	128	50.0
ST3 Training	8000	117	4.3	2.4	60.5	6	6	206	50.0
ST3 Test	2000	117	4.2	2.4	59.0	6	6	192	50.0

The selected datasets provide different challenges to the two classification approaches. Note that the *CSLOG* and synthetic datasets are structurally heterogeneous, while the hospital and *CRM* datasets are structurally homogeneous. In the case of the hospital dataset, the node repetitions occur at all levels except at the lowest level, leading to performance bottle-neck for both *DSMC* and *XRules* in their frequent subtree mining process, even at a minimum support value of 10%. The results and discussions of the *CSLOG* and synthetic dataset are presented in Section 5.3. The experiments on hospital and *CRM* datasets are presented in Section 5.4.

5.3 Experimental Results on Structurally Heterogeneous Data

Section 5.3.1 presents the general results of the application of the *DSMC* method on *CS1-2*. The results on *CS2-3* and *CS3-1* are not shown here due to similar trends (they can be found in Appendix A). The number of rules generated by the two methods and the rules triggered by two specific test instances are analysed in Section 5.3.2. The performances of *DSMC* and *XRules* on majority and minority class are examined in Section 5.3.3. Finally, Section 5.3.4 discusses the experimental results on synthetic datasets.

5.3.1 General Comparison

The evaluation results are organised into several subsections based on classification approaches, frequent pattern types, subtree types, rule strength types and support thresholds. The last subsection reports a statistics performed on the data.

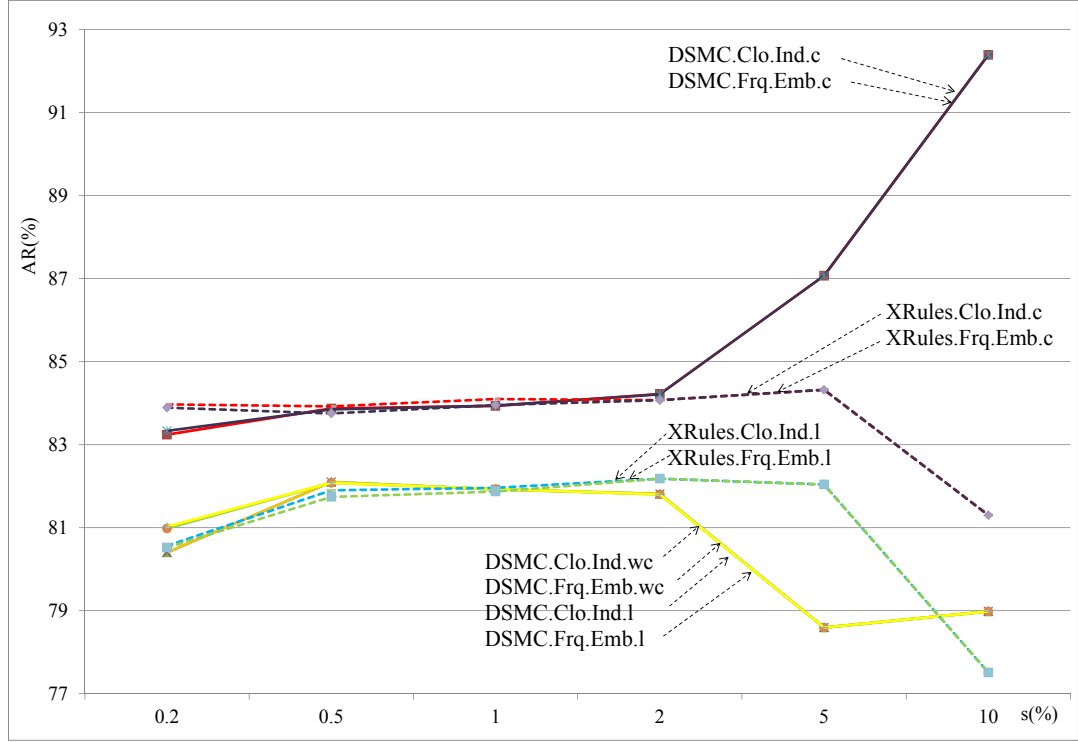


Figure 5.8: Accuracy rates of dataset *CS1-2* where $(w)c=50\%$ or $l=1$.

DSMC vs XRules

The accuracy and coverage rates of the two methods are examined for different combinations of frequent pattern type, subtree type, rule strength measure/threshold and minimum support value. Fig. 5.8 and 5.9 show the accuracy rates when the rule strength is $(w)c=50\%$ or $l=1$, and $(w)c=90\%$ or $l=10$, respectively.

Only similar configurations of the two methods are compared, e.g. *DSMC.Frq.Emb.c* is compared with *XRULES.Frq.Emb.c*. It is seen from Fig. 5.8 that the accuracy rates of *DSMC* and *XRULES* are much different in most settings. The *DSMC*'s accuracy rates are higher than those of *XRULES* when confidence is used for rule strength and the support threshold is higher than 2% (note that the results of the weighted confidence option of *XRULES* are not shown because they are equal to those of confidence option).

For $(w)c=90\%$ or $l=10$ (see Fig. 5.9), *DSMC* and *XRULES* have similar accuracy rates when the confidence option is used. If weight confidence option is used, the *DSMC* method has lower accuracy rates. When the likelihood option is used, the *XRULES* method is better than its counterpart in case $s = 2\%$ or 5% . *XRULES* performs no better than *DSMC* in the remaining cases.

In terms of coverage rate, *XRULES* gives better results than *DSMC* for all configurations at $(w)c=50\%$ or $l=1$ and $(w)c=90\%$ or $l=10$ (see Fig. 5.10 and 5.11, respectively).

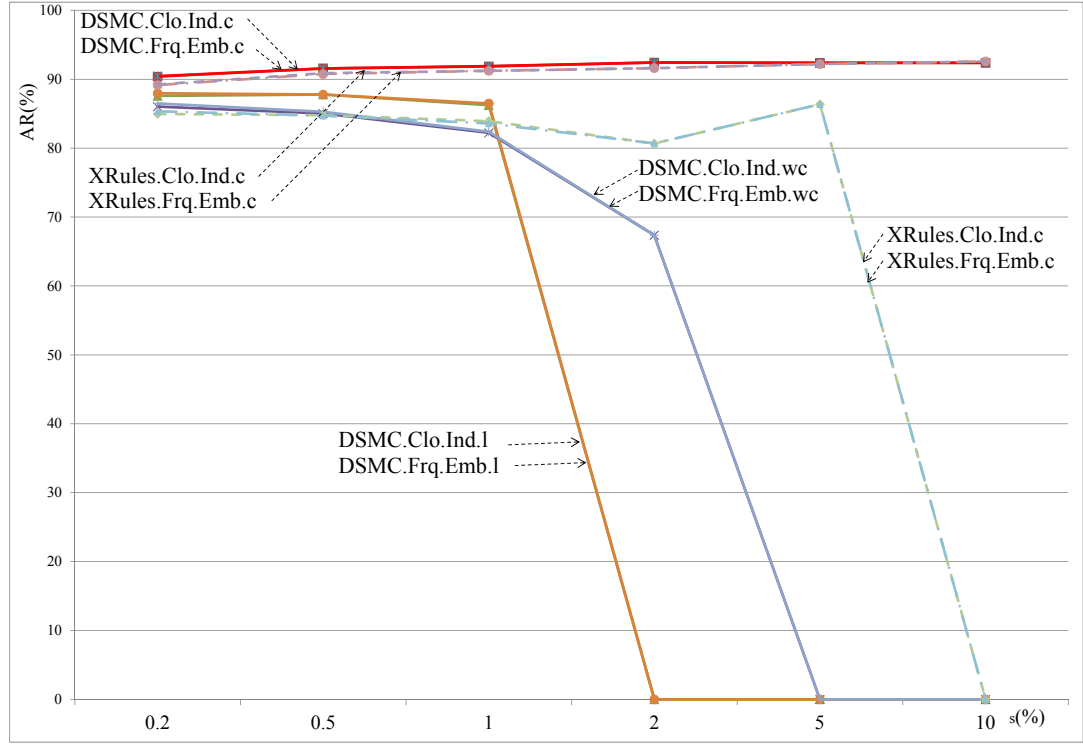


Figure 5.9: Accuracy rates of dataset *CSI-2* where $(w)c=90\%$ or $l=10$.

Frequent Pattern Types

The frequent subtree and closed subtree options are evaluated using various configurations. However, the minimum support values are fixed at $s=0.2\%$ or $s=2\%$ and the rule strength thresholds are fixed at $(w)c=50\%$ or $l=1$. The results in Fig. 5.12 show that the accuracy rates of the closed subtree option are slightly higher than those of the frequent subtree option for both minimum support values. In terms of coverage rate, the frequent subtree option yields slightly better results than the closed subtree option when $s=0.2\%$ (see Fig. 5.13).

Subtree Types

Due to the large number of possible configurations, the frequent subtree and confidence options are selected as representatives of the frequent pattern type and rule strength type, respectively. For example, the results of *DSMC.Frq.Emb.c* represent for the results of *DSMC.Emb*. For *XRULES.Ind*, the closed subtree option is selected instead of frequent subtree option. The accuracy and coverage rates are shown in Fig. 5.14 and 5.15, respectively. As shown in Fig. 5.14 and 5.15, the accuracy and coverage rates are not affected by the subtree types, except that there is a slight difference (less than 1%) among the values for *XRULES* when $s \leq 2\%$.

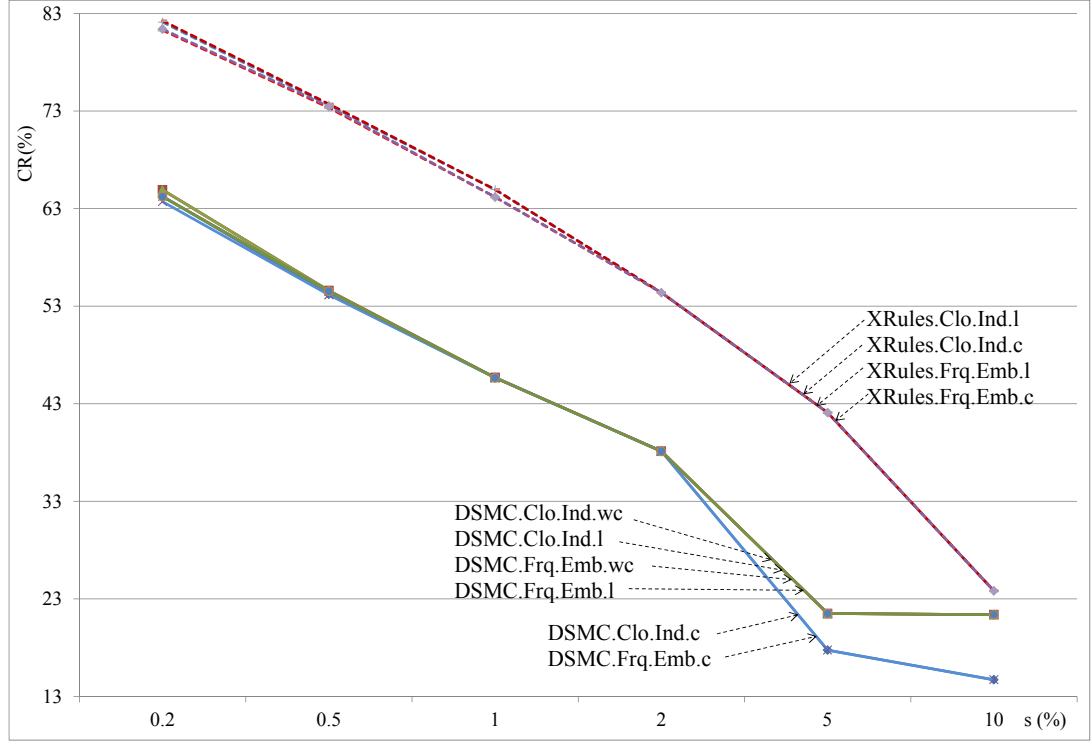


Figure 5.10: Coverage rates of dataset *CSI-2* where $(w)c=50\%$ or $l=1$.

Rule Strength Types

The confidence, weighted confidence, and likelihood options are evaluated based on the averaged accuracy and coverage rates of different combinations of methods, frequent pattern types and subtree types over all selected support and rule strength thresholds. Fig. 5.16 and 5.17 show that the accuracy and coverage rates of the confidence option are the highest and those of weighted confidence option come in second. Note that for *XRULES*, the weighted confidence option has the same performance as the confidence option.

As shown in Fig. 5.8 and 5.9, when $s < 2\%$, increasing the rule strength thresholds improves the accuracy rates. On the other hand, the coverage rates drop when the rule strength threshold rises (see Fig. 5.10 and 5.11).

In another experiment, the performance of *DSMC* is compared with different types of rule strengths (see Section 2.3.14 for more detail) on the *CSI-2* and hospital datasets. The closed, induced subtree option with a minimum support of 0.5% is used for the *CSI-2* dataset and the closed, embedded subtree option with a minimum support of 10% is used for the hospital dataset.

The results of the experiment are displayed in Table 5.6. In the *CSI-2* dataset, it is seen that the best combination of accuracy and coverage rates achieved are approximately above 83% and 54% (those printed in bold font), respectively. These results are obtained when confidence = 50%, weighted confidence = 50%, cosine = 0.05, convic-

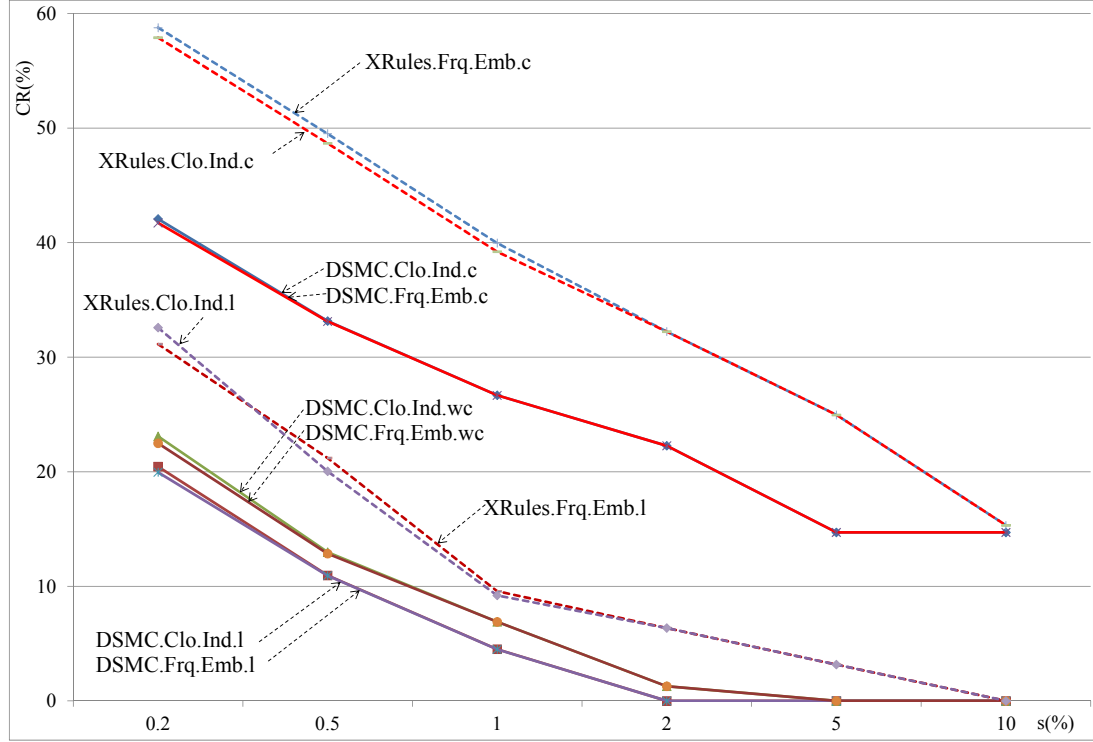


Figure 5.11: Coverage rates of dataset *CS1-2* where $(w)c=90\%$ or $l=10$.

$tion = 1$, and $Laplace = 0.5$. In the hospital dataset, the above rule strength measures also produce the best results in terms of combinations of accuracy and coverage rates ($AR \geq 70\%$ and $CR = 100\%$).

In summary, from the experiments it was shown that different types of interestingness measures can be used for the *DSMC* method, and the best combinations of accuracy and coverage rates are achieved by the confidence, weighted confidence, conviction and Laplace options. Note that the differences in terms of best combinations of accuracy and coverage rates between the evaluated interestingness measures are negligible, except for the *Yule's Q*, *GINI* and *lift* measures where there is a drop of either accuracy or coverage rate. In practice, the confidence measure seems to be a preferred option due to its simplicity and straightforward interpretation.

Supports

It can be seen from Fig. 5.11 that the higher the support threshold is set, the lower the coverage rate is obtained. This can be explained by the fact that increasing the minimum support value leads to fewer frequent subtree patterns and thus reduced rules' coverage. Fig. 5.14 shows that increasing the support value does not affect the accuracy rates for both *DSMC* and *XRules* when $s < 2\%$. The results in Fig. 5.15 show that the support values are inversely proportional with the coverage rates for both methods.

Table 5.6: The accuracy and coverage rates of *DSMC* when different rule strength types are applied to the *CS1-2* and *Hospital* datasets. Note that Th=Threshold.

		CS1-2		Hospital	
Measure	TH	AR	CR	AR	CR
Confidence	0.5	83.87	54.23	70.89	100.00
	0.9	91.53	33.33	72.31	82.28
Weighted Confidence	0.5	83.09	54.60	70.89	100.00
	0.9	85.03	12.98	68.66	84.84
Cosine	0.005	82.22	54.60	69.62	100.00
	0.01	82.22	54.60	69.62	100.00
	0.05	83.15	54.56	69.62	100.00
	0.1	82.57	43.74	69.62	100.00
Yule's Q	0.2	82.83	54.31	37.97	100.00
	0.5	81.83	47.09	37.97	100.00
	0.8	85.03	12.98	37.97	100.00
GINI	0.5	63.39	54.60	-	-
	0.9	63.39	54.60	-	-
	0.001	39.92	37.56	37.97	100.00
	0.005	23.22	22.78	37.97	100.00
Lift	2	67.46	14.77	53.85	49.37
	5	-	0.00	-	0.00
	10	-	0.00	-	0.00
Conviction	1	83.09	54.60	70.89	100.00
	5	90.65	12.85	74.84	83.54
	10	91.04	4.82	73.33	75.95
Laplace	0.5	83.87	54.23	70.89	100.00
	0.7	88.65	46.01	73.42	100.00
	0.9	91.66	30.41	75.44	72.15
Leverage	0.00	82.22	54.60	70.89	100.00
	0.01	78.98	21.38	70.89	100.00
	0.05	-	0.00	70.89	100.00
Jaccard	0.00	82.22	54.60	70.89	100.00
	0.01	81.92	45.69	70.89	100.00
	0.1	92.39	14.71	70.89	100.00
	0.5	-	0.00	62.03	100.00
Pearson	0.00	82.22	54.60	37.97	100.00
	0.1	78.85	25.15	37.97	100.00

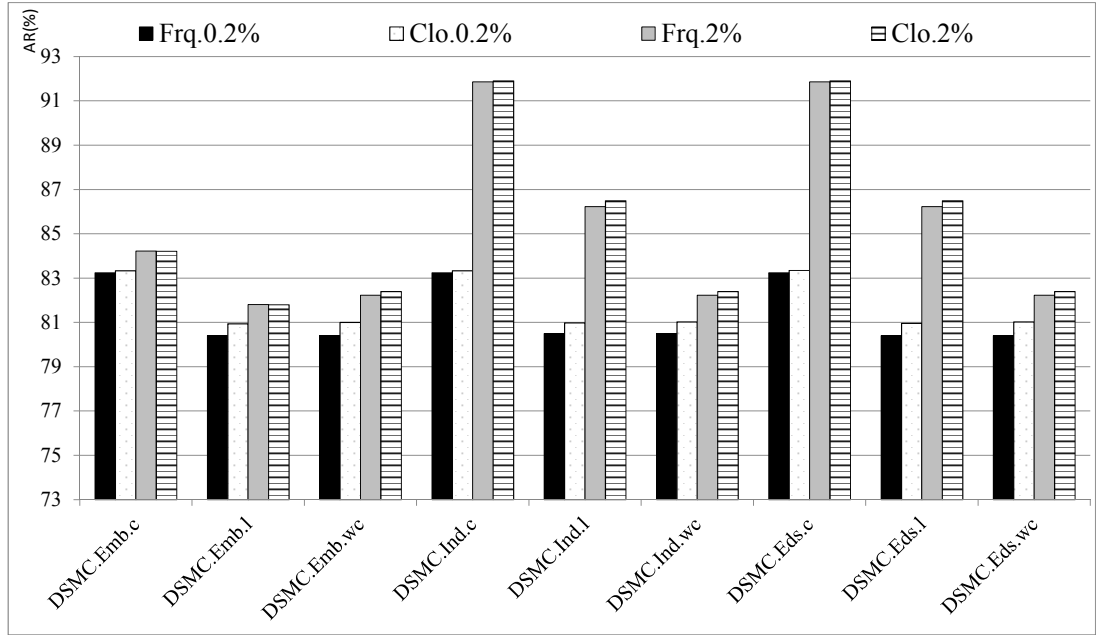


Figure 5.12: Comparison of accuracy rates based on frequent pattern types in the *CSI-2* dataset.

Evaluation based on Cross-Validation

The experiment setting for this evaluation is $\varepsilon = 1\%$, $c = 50\%$, and the closed, induced subtree option is used. The dataset is divided into ten partitions, each of which is denoted as f_i , with $i \in [0..9]$. The experiments run ten times and in each run, one partition is used as test data, and the remaining partitions are combined for training data.

Table 5.7 shows the evaluation results. A paired student's t-test statistics is then applied to these results with a significance level of 0.05. The p -value of the two-tailed test with a degree of freedom of 9 is 0.87, which is greater than the critical value. This means that the null hypothesis cannot be rejected, which means that there is no statistical difference between the two methods.

Table 5.7: The accuracy rates on ten-fold cross-validation of the *CSI-2* dataset.

	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	Mean
<i>DSMC</i>	85.8	81.8	84.0	82.9	88.4	83.6	83.3	87.6	85.3	88.0	85.07
<i>XRules</i>	83.5	84.4	86.4	83.6	89.4	82.9	82.2	88.1	85.0	86.1	85.16

5.3.2 Analysis of the Rule Set

Table 5.8 shows the number of rules of each variation of the two approaches on three pairs of training and test set. It is noticeable that the number of rules for the closed subtree option is much less than that of the frequent subtree option when $\varepsilon=0.2\%$,

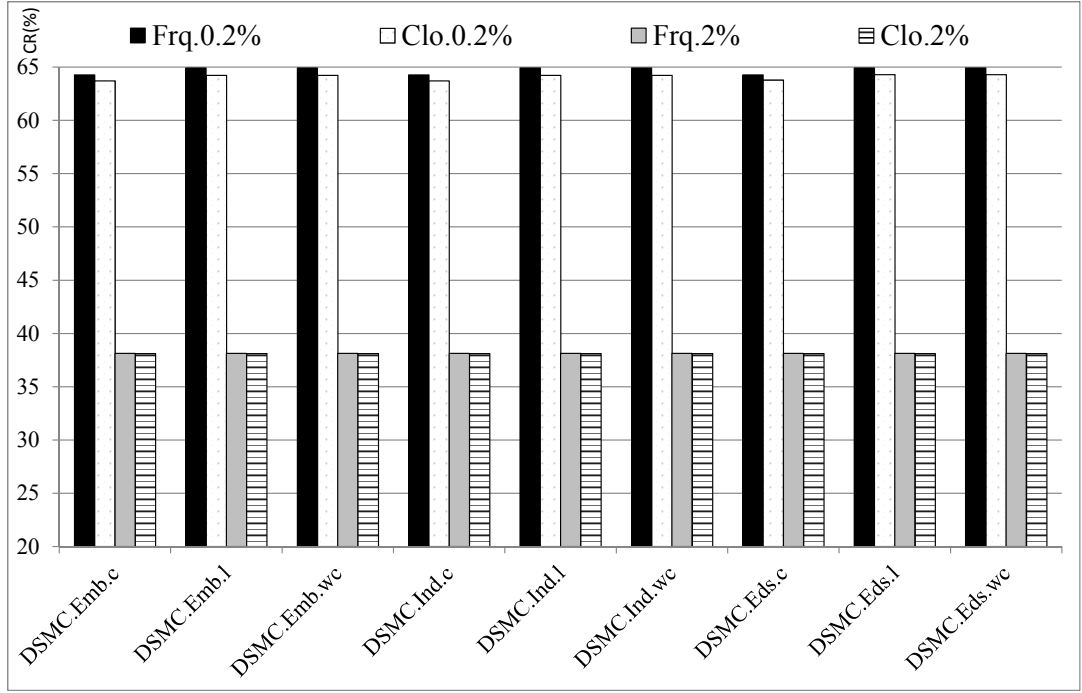


Figure 5.13: Comparison of coverage rates based on frequent pattern types in the *CSI-2* dataset.

though the difference does not affect much on the accuracy and coverage rate. And as expected, the number of rules of the embedded subtree option is always higher than that of induced ones.

The fact that *Eds* variations produce the highest number of rules can be understood by their inclusion of both embedded subtree and disconnected subtree patterns.

There is almost no difference in the number of rules found at support thresholds $\mathfrak{s}=5\%$ and $\mathfrak{s}=10\%$.

It is seen that *XRules* enumerates significantly more rules at lower support thresholds since its frequent/closed subtrees are not constrained by position like those of *DSMC* (note that the *XRules*'s implementation limits the number of frequent subtrees to be less than 50,000). This explains the high coverage rate of *XRules*, as can be seen from the results of *CSI-2* in Fig. 5.10 and 5.11.

Generally speaking, the two methods have comparable accuracy rates, while there is a larger degradation in coverage rate when the position constraint is imposed as part of the *DSMC* approach.

Analysing the rules triggered in a single test case where *DSMC* and *XRules* give contradictory predictions helps explain some of the results in Section 5.3.1. Given *CSI-2* as the training and test data, the minimum support and confidence are both set at 50%, and closed embedded subtrees are selected as rule antecedents, each of the following cases shows each method in its favoured result.

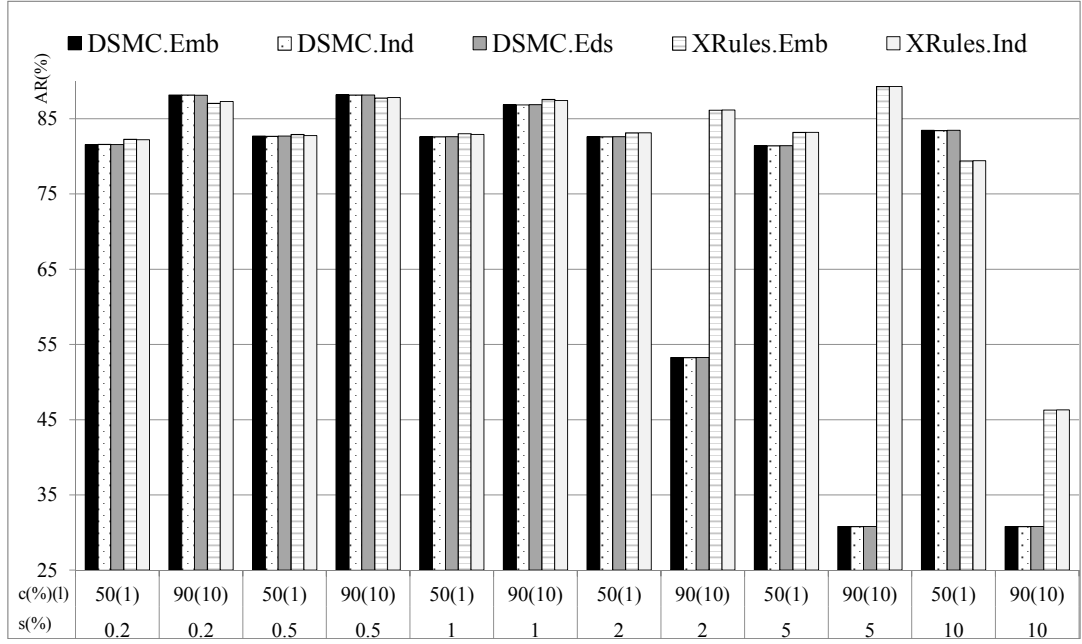

 Figure 5.14: Comparison of accuracy rates based on subtree types in the *CS1-2* dataset.

Table 5.8: Number of rules.

Dataset	CS1-2						CS2-3						CS3-1					
s (%)	0.2	0.5	1	2	5	10	0.2	0.5	1	2	5	10	0.2	0.5	1	2	5	10
<i>DSMC.Frq.Emb</i>	8298	316	75	26	4	2	11907	356	83	27	4	2	13775	357	100	26	4	2
<i>DSMC.Clo.Emb</i>	853	230	69	25	4	2	752	230	72	26	4	2	835	250	87	25	3	2
<i>DSMC.Frq.Ind</i>	1973	290	75	26	4	2	1409	316	83	27	4	2	7303	331	100	26	4	2
<i>DSMC.Clo.Ind</i>	830	229	69	25	4	2	728	228	72	26	4	2	804	249	87	25	3	2
<i>DSMC.Frq.Eds</i>	9830	349	79	26	4	2	12407	401	89	28	4	2	20848	411	112	27	4	2
<i>DSMC.Clo.Eds</i>	955	248	73	25	4	2	841	251	76	27	4	2	972	278	94	26	3	2
<i>XRULES.Frq.Emb</i>	50000	1764	367	223	19	7	50000	3285	371	102	21	3	50000	2671	401	114	20	3
<i>XRULES.Clo.Ind</i>	7434	1193	296	98	16	3	7699	1063	289	97	18	3	6093	1319	322	109	17	3

DSMC+:

Table 5.9 shows the rules triggered by *DSMC* and *XRULES* for a test instance (t_x) ‘0 1 16 254 1957 -1 -1 -1 -1’ with class label C_1 . Note that X_{99} is the last column of the tabular format of the training data, which represents the instances’ class labels. In particular, $X_{99} = 0$, $X_{99} = 1$ corresponds to class C_1 , C_2 , respectively. In the *DSMC* method, $\rho_{C_1}(t_x) > \rho_{C_2}(t_x)$, thus t_x is classified as C_1 . For *XRULES*, t_x is classified as C_2 . In this example, *DSMC* correctly classifies instance t_x whereas *XRULES* does not.

If the positional information is removed from the antecedents of the *DSMC* rules, three corresponding pairs of rules are obtained: the first, second, and third rule of *DSMC* corresponds to the first, second, and third rule of *XRULES*, respectively. The first rule of each method predicts C_2 . The second and third rules of *DSMC* predict C_1 , whereas their counterparts in *XRULES* predict C_2 . It can be inferred that due to the last two rules of *DSMC*, the combined rule strength of the *DSMC* method on class C_1 is

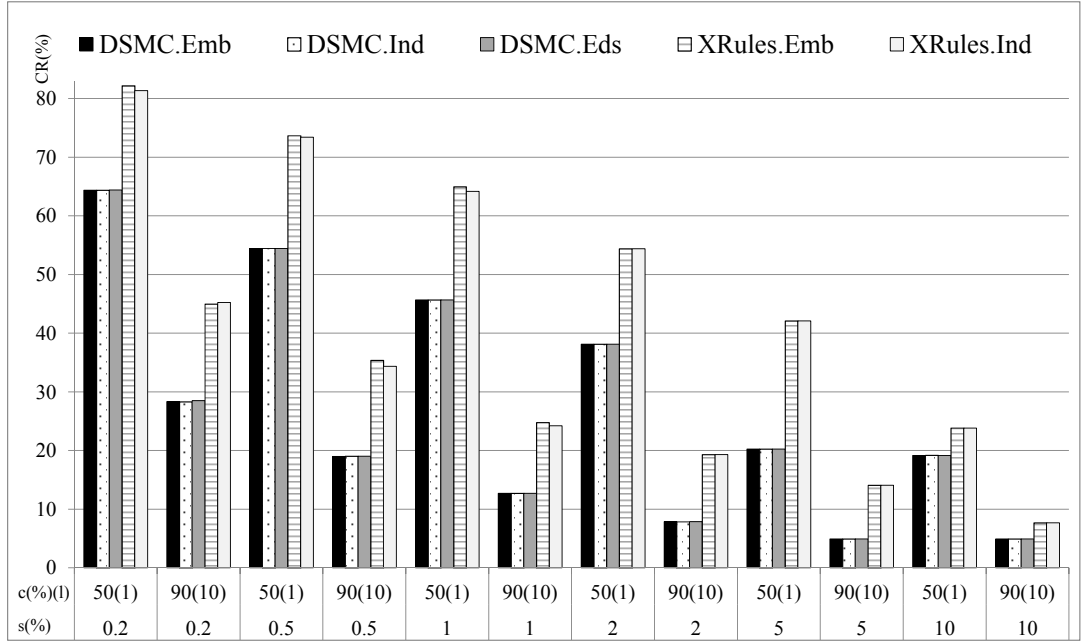


Figure 5.15: Comparison of coverage rates based on subtree types in the *CSI-2* dataset.

Table 5.9: *DSMC+*.

	Rule	support	confidence	$\rho_{c,C_1}(t_x)$	$\rho_{c,C_2}(t_x)$
DSMC	$X_1=1 \Rightarrow X_{99}=1 (C_2)$	236	0.52	0.54	0.52
	$X_1=1, X_2=16 \Rightarrow X_{99}=0 (C_1)$	12	0.5		
	$X_2=16 \Rightarrow X_{99}=0 (C_1)$	16	0.57		
XRules	$1 \Rightarrow C_2$	253	0.53	0	0.54
	$1, 16 \Rightarrow C_2$	30	0.57		
	$16 \Rightarrow C_2$	36	0.51		

higher than that of class C_2 , which leads the correct prediction of t_x . It is noticeable that the last two rules of *DSMC* have lower supports than their counterparts of *XRules* (12, 16 in comparison with 30, 36, respectively). The behaviour shown above can be observed in a majority of test instances where *DSMC* gives accurate prediction while *XRules* produces incorrect result. Although the position-constrained-based rules have lower support than traditional subtree-based rules, they have better prediction power.

***XRules+*:**

Table 5.10 shows the rules triggered by *DSMC* and *XRules* for the test instance ‘0 1 8 -1 -1 6 81 -1 -1’. In the *DSMC* approach, only one rule is triggered $X_1=1 \Rightarrow X_{99}=1$ (predicting class C_2). In the *XRules* approach, four rules are generated $6 \Rightarrow C_1$, $81 \Rightarrow C_1$, $6, 81 \Rightarrow C_1$, $1 \Rightarrow C_2$. *XRules* accurately classifies the instance while *DSMC* does not. It should be noted that the additional rules found by *XRules* do not occur sufficiently enough at the same position to be found by *DSMC* and average out

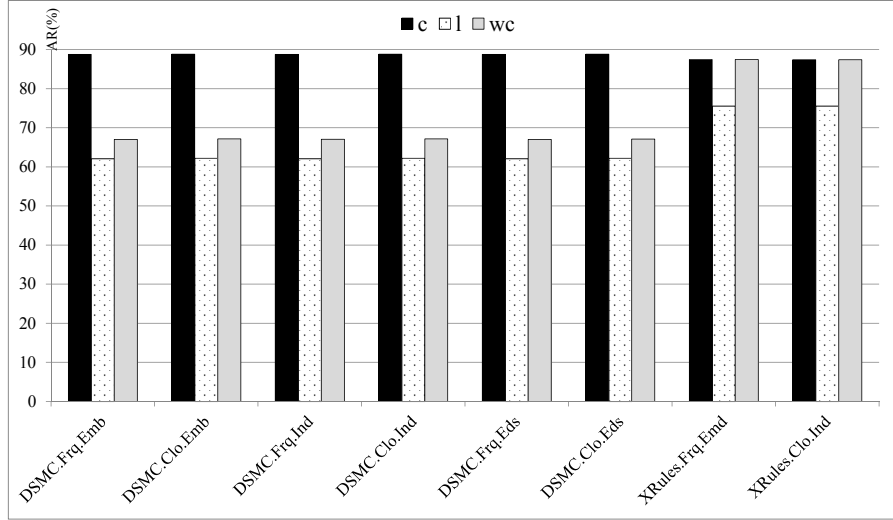


Figure 5.16: Comparison of accuracy rates based on rule strength measures in the CSI-2 dataset.

Table 5.10: *XRules*+

	Rule	support	confidence	$\rho_{c,C_1}(t_y)$	$\rho_{c,C_2}(t_y)$
DSMC	$X_1=1 \Rightarrow X_{99}=1 (C_2)$	236	0.52	0	0.52
	$1 \Rightarrow C_2$	253	0.53	0.63	0.53
XRules	$6 \Rightarrow C_1$	208	0.66		
	$81 \Rightarrow C_1$	43	0.61		
	$6, 81 \Rightarrow C_1$	17	0.61		

the prediction towards C_1 . However, if the support threshold is lowered to a certain value, these extra rules can be found by the *DSMC*, but in different position-constrained variants of the same rule.

Examining the set of rules triggered by several test cases, it is seen that the number of triggered rules of *DSMC* is often smaller than that of *XRules*. This could be attributed to the position constraint being applied to the frequent subtree mining process. In cases where there is a great difference between the numbers of triggered rules among the two methods, *XRules* often gives a more precise prediction.

Several test cases are examined where *XRules* rules that do not have their counterparts in *DSMC* (*DSMC*'s rules that when mapped to tree format are similar to that of *XRules*) are removed from the rule set. Fig. 5.18 shows that the number of correct predictions (on instances that are covered by both methods) of *DSMC* is higher on average than that of *XRules*.

The reason for the better predictive capability of *DSMC* in this case might be that the position-constrained patterns better reflect the class of the instance. The results encourage us to combine *DSMC* and *XRules* to achieve a better accuracy rates without a reduction of coverage rates. The following strategies are proposed for combining

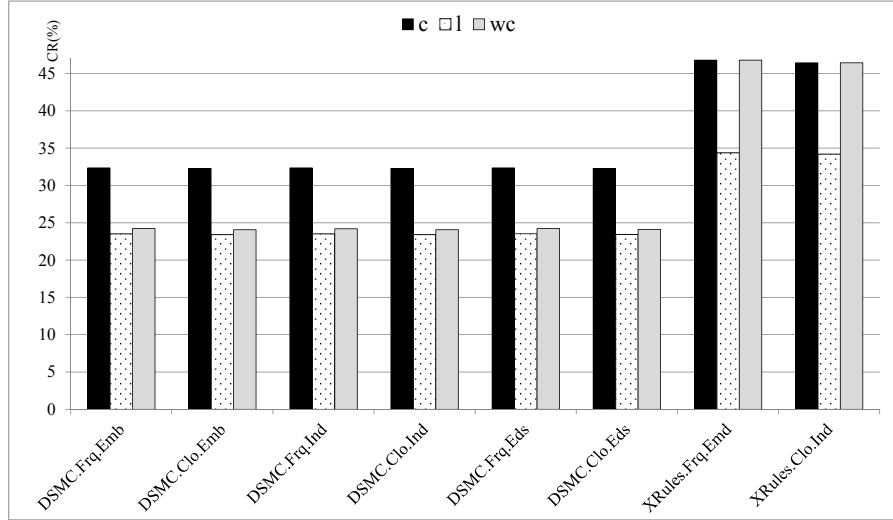


Figure 5.17: Comparison of coverage rates based on rule strength measures in the *CSI-2* dataset.

the rule sets of the two methods and evaluated them on the *CSI-2* dataset using closed induced subtree with a minimum support of 0.5% and a minimum confidence of 50%.

1. The final rule set includes both *DSMC* and *XRULES*'s rules. For each test instance, the *DSMC*'s rules are examined first, if none of the rules are triggered, then *XRULES*'s rules are examined. The accuracy rate achieved for this rule set is 83.26%.
2. The final rule set includes both *DSMC* and *XRULES*'s rules. For each test instance, both rule sets are examined and the rule set that has more triggered rules is selected. The achieved accuracy rate is 83.47%.
3. The final rule set includes rules from either *DSMC* or *XRULES*: if two rules' antecedents share the same subtree, the rule that has a higher confidence is selected. The achieved accuracy rate is 83.48%.

Since the accuracy rates of *DSMC* and *XRULES* for the original settings are 83.85% and 83.75%, respectively, the proposed strategies for combining the two methods showed no improvements. This is probably due to the similar performance of *DSMC* and *XRULES* in those instances that both methods cover. However, there is a slight improvement in terms of coverage rates. The results for *DSMC* and *XRULES* alone are 54.16% and 73.34%, respectively, but increase to 73.46% for the three strategies described above.

The experiments performed in this section show that with the same support threshold, traditional approaches outperform *DSMC* in terms of coverage rates due to additional subtree patterns that are frequent when they are not constrained by positions.

However, it was demonstrated in (Hadzic et al., 2015) that for many datasets, more subtree patterns might be enumerated by the position constrained approach overall. This is because subtree patterns could be extracted at a much lower support threshold in comparison to traditional approaches. At lower support thresholds, position-constrained variants of traditional subtrees would also become frequent, thus more rules could be discovered.

In the experiment of the first strategy in combining position-constrained and traditional rules, the minimum support of *DSMC* is changed to 0.1% (previously 0.5%) while other settings are the same. The accuracy and coverage rates achieved are 83.66% and 76.08%, respectively. Though the accuracy rate of the new setting does not clearly improve over each method alone, its coverage rate is 2.74% and 21.92% higher than *XRules* and *DSMC*, respectively. This result shows the potential of using a lower minimum support for *DSMC* in combination with a higher minimum support for the traditional approach. More studies are needed in finding a more effective strategy to combine the two approaches and to reduce the computational cost related to the rule matching/selection.

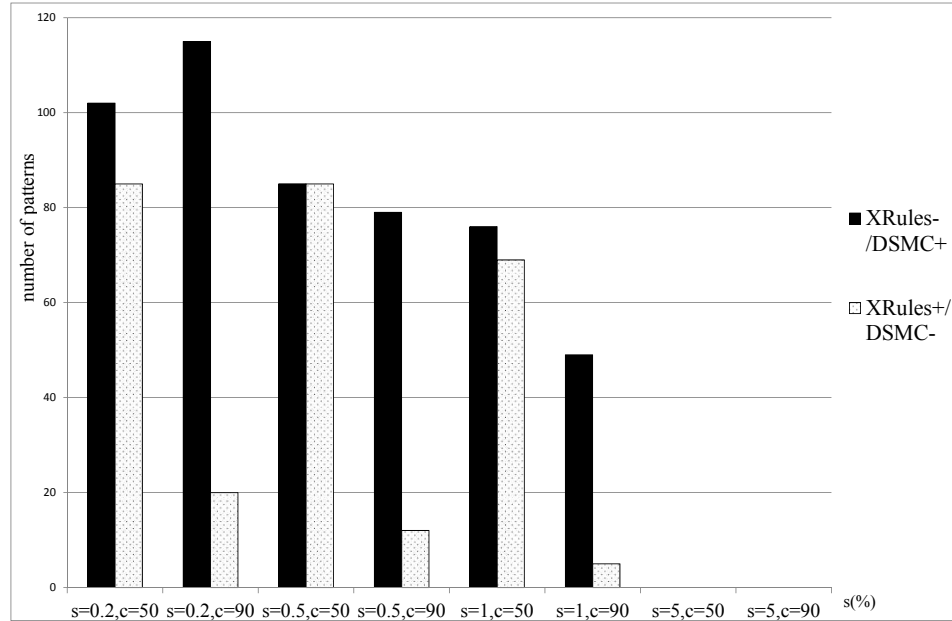


Figure 5.18: The number of instances which one method predicts correctly whereas the other wrong (based on similar rules).

5.3.3 Comparison on Imbalanced Classes

Table 5.11 shows the accuracy and coverage rate of different techniques when applied to the three pairs of training and test sets with the setting of frequent embedded subtree

Table 5.11: Accuracy and coverage rates for individual classes in the *CSLOG* dataset.

	Method	AR(%) for ς (%)											
		0.2		0.5		1		2		5		10	
Accuracy Rate	CS1-2	DSMC	88.06 70.06	88.09 71.65	88.42 71.14	90.51 59.30	92.05 62.44	100.00 0					
		XRules	91.15 61.01	90.99 62.56	90.70 65.36	89.90 67.31	91.27 62.20	88.64 56.36					
		XRulesS	90.30 65.20	90.80 65.83	91.02 65.11	92.20 52.11	92.05 62.44	100.00 0					
	CS2-3	DSMC	88.86 69.77	89.26 68.71	90.76 64.90	91.87 56.05	93.06 62.33	100.00 0					
		XRules	91.82 62.79	91.14 67.17	90.72 67.22	90.17 68.70	92.84 59.60	89.81 55.58					
		XRulesS	90.77 66.42	91.17 65.03	92.82 58.78	93.22 51.27	93.06 62.33	100.00 0					
	CS3-1	DSMC	90.09 67.59	89.91 70.89	86.57 73.06	88.52 64.35	82.25 70.74	82.37 70.65					
		XRules	91.22 63.85	90.86 66.23	90.22 68.79	89.66 69.71	93.79 56.86	90.99 52.53					
		XRulesS	90.29 68.59	90.36 70.72	91.55 64.54	93.93 52.31	92.84 41.80	93.05 40.65					
Coverage Rate	CS1-2	DSMC	60.91 75.64	52.22 61.06	43.78 52.16	39.43 33.79	19.12 13.10	17.60 4.92					
		XRules	80.99 85.89	71.60 80.44	62.20 74.27	52.25 61.65	41.45 44.22	23.84 23.77					
	CS2-3	DSMC	60.99 73.97	52.54 58.84	45.87 48.16	39.47 34.93	19.28 12.40	17.94 4.67					
		XRules	81.37 87.43	70.12 80.65	60.10 73.97	52.50 64.68	41.92 41.99	23.91 22.91					
	CS3-1	DSMC	61.60 71.87	51.75 60.75	46.29 47.86	39.05 30.89	21.94 15.85	21.91 15.80					
		XRules	80.10 83.89	71.60 77.88	61.21 72.17	52.24 60.24	40.84 36.39	23.43 20.18					

and $\varsigma=50\%$. The performance of the majority and minority class are shown on the left and the right of each cell, respectively. It can be seen that *XRules* has better accuracy rates in 11/18 cases for the majority class and *DSMC* has better accuracy rates in 12/18 cases for the minority class.

Since *DSMC* has lower coverage rate than *XRules*, it is important to measure accuracy of *XRules* relative to only those instances that *DSMC* covered. The rows labelled as *XRulesS* correspond to the results when *XRules* is evaluated on only those instances that are covered by the *DSMC* method. Note that *XRulesS* typically covered all instances that *DSMC* covers except for a few instances at $\varsigma=0.2\%$.

It is seen that *XRulesS* has better accuracy rates in all cases for the majority class, while *DSMC* has better accuracy rates in 17/18 cases for the minority class.

Table 5.12 adds more details to this comparison by showing the instances that *DSMC* approach predicted correctly but *XRulesS* misclassified (called *DSMC+* *XRulesS*- and instances that *DSMC* approach misclassified but *XRulesS* classified correctly (called *DSMC*- *XRulesS*+). The results of the majority and minority class are shown on the left and right, respectively of each cell in the table.

It is observed that the number of misclassifications of *DSMC* is lower than that of *XRulesS* in 17/18 cases for the minority class and the result of *XRulesS* is better than that of *DSMC* in all cases for the majority class. The results confirm that *DSMC* has an advantage in accuracy rate for the minority class and it is the other way around for the majority class in the *CSLOG* dataset.

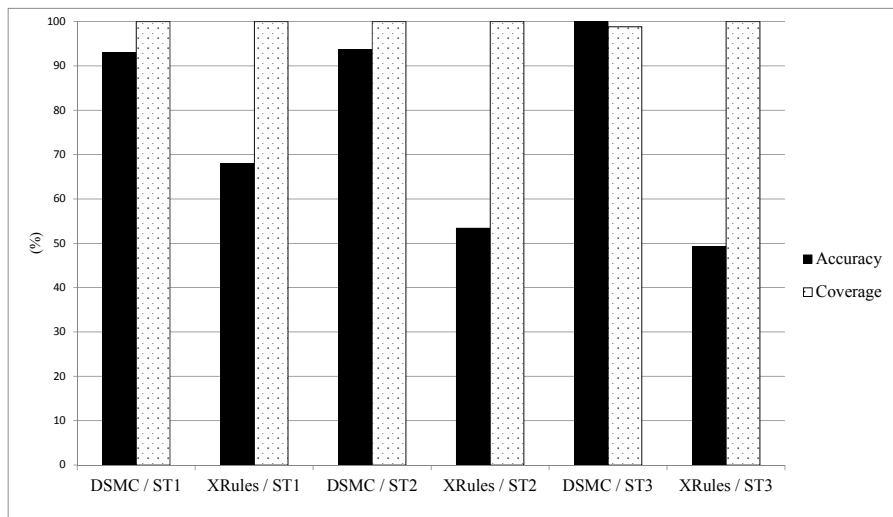
Table 5.12: Instances correctly(+)/incorrectly(-) classified by *DSMC* and *XRulesS*.

		Number of instances for \mathfrak{s} (%)							
		0.2	0.5	1	2	5	10		
CS12	<i>DSMC</i> + <i>XRulesS</i> -	16 72	07 62	12 63	21 53	0 0	0 0		
	<i>DSMC</i> - <i>XRulesS</i> +	93 10	88 02	77 10	59 12	0 0	0 0		
CS23	<i>DSMC</i> + <i>XRulesS</i> -	20 67	10 57	06 61	16 42	0 0	0 0		
	<i>DSMC</i> - <i>XRulesS</i> +	86 23	85 18	61 08	47 12	0 0	0 0		
CS31	<i>DSMC</i> + <i>XRulesS</i> -	47 47	16 22	00 80	00 73	0 90	0 93		
	<i>DSMC</i> - <i>XRulesS</i> +	49 62	30 20	141 0	129 0	142 0	143 0		

5.3.4 Synthetic Datasets

In this section, the accuracy and coverage rates of *DSMC* and *XRules* on three synthetic datasets at minimum supports of 1% and 2% are examined. Both methods use closed, induced subtrees as antecedents of classification rules and a confidence threshold of 50%. Other settings of frequent pattern, subtree, and rule strength type result in similar results and are not presented here.

Fig. 5.19 and 5.20 show that the accuracy rates produced by *DSMC* is significantly higher than that of *XRules* in all datasets; especially in *ST3*, the accuracy rates of *DSMC* are twice of that of *XRules* for both minimum support values. When the minimum support is set at 1%, there are no differences between the coverage rates of the two methods. If the threshold is 2%, the coverage rates of *DSMC* are 15% and 36% lower than that of *XRules* in *ST2* and *ST3*, respectively. The reason could be that there are fewer position-constrained subtrees found at higher support thresholds.

Figure 5.19: Accuracy and coverage rates of synthetic datasets with $\mathfrak{s} = 1\%$.

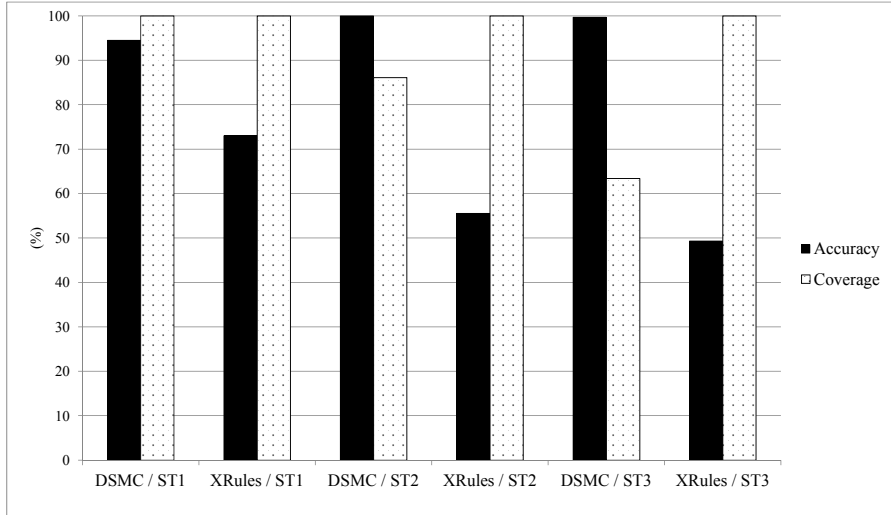


Figure 5.20: Accuracy and coverage rates of synthetic datasets with $\epsilon = 2\%$.

5.4 Experimental Results on Structurally Homogeneous Data

As earlier mentioned, since the hospital and *CRM* datasets are structurally homogeneous, a challenge they pose for frequent subtree mining task lies in the large number of repetitions of sibling nodes/substructures which could lead to exponential running time and memory usage. Thus closed subtree option is preferred to frequent subtree option in this dataset. Also due to the homogeneity of the data, there is no difference in the resulting closed induced and closed embedded subtrees and no disconnected subtrees were found. Hence, all variations of *DSMC.Clo.Emb* and *DSMC.Clo.Eds* are not shown as they are the same as *DSMC.Clo.Ind*.

It is observed that *XRules.Clo.Ind* is not able to complete at any support threshold. The data is then modified so that all repetitions in sibling nodes are eliminated. In case the dataset is modified, the performances of the two methods are evaluated based on the maximum accuracy or coverage rates of each method.

The results of *XRules* using the weighted confidence option are not shown since they are identical to those using confidence option. The number of rules found by the two approaches for the two dataset is presented in Table 5.13. It shows that the number of rules of *XRules* is higher than that of *DSMC* but the difference is subtle. This can be explained by the fact that by uniquely relabelling the sibling nodes, the number of subtree patterns found has reduced to a level that is comparable to that of *DSMC*.

Note that the statistical test and the ten-fold cross-validation were not performed on the (original) hospital and *CRM* datasets because *XRules* crashed early at the high support threshold (e.g. 60% for the hospital dataset), thus only few patterns were found.

Table 5.13: Number of rules of the hospital and *CRM* dataset.

Dataset	Hospital			CRM			
s (%)	5	10	50	1	5	10	50
DSMC.Clo.Ind	47945	35540	143	7443	1045	408	16
XRules.SR.Clo.Ind	-	36805	216	10103	1410	561	19

5.4.1 Hospital Dataset

Table 5.14 and 5.15 display the accuracy and coverage rates of the subset of the hospital dataset. Note that highest values per setting are shown in bold. In this dataset, the support threshold below 5% is not examined as the number of instances is quite small in the training data (160 instances). It is seen that the accuracy rates of *DSMC* are higher than *XRules* when $s=5\%$, while for other values the opposite is the case. The confidence option gives better accuracy rates than the other two options at $(w)c=50\%$ or $l=1$.

For both approaches, increasing the rule strength thresholds leads to mixed results in accuracy rate. Mixed results are also found when increasing minimum support values. At the highest minimum support ($s=50\%$), (weighted) confidence ($(w)c=90\%$) or likelihood threshold ($l=10$), the rule sets reduce into empty sets. It is noticeable that at $s=5\%$ and the option is *XRules.SR.Clo.Ind*, the program fails to terminate after one day running. At $(w)c=50\%$ or $l=1$, the resulting coverage rates for both *DSMC* and *XRules* are 100%.

From the coverage rate results of Table 5.15, it is shown that increasing the rule strength thresholds results in a decrease in coverage rate. Overall, the *DSMC* method gives better maximum coverage rates and *XRules* gives better accuracy rates where it obtained three of the best accuracy values compared to two of *DSMC*.

Table 5.14: Accuracy rates of the hospital dataset.

s (%)	5		10		50	
$(w)c(\%)/l$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
DSMC.Clo.Ind. c	72.15	56.52	70.89	64.62	68.35	0
DSMC.Clo.Ind. l	67.09	68.12	69.62	69.84	70.89	0
DSMC.Clo.Ind. wc	70.89	57.75	70.89	65.67	70.89	0
XRules.SR.Clo.Ind. c	-	-	72.15	78.57	74.68	0
XRules.SR.Clo.Ind. l	-	-	68.35	68.75	70.89	0

5.4.2 CRM Dataset

Table 5.16 and 5.17 show the accuracy and coverage rates, respectively, when applying *DSMC* and *XRules* to the *CRM* dataset. Note that highest values per setting are shown

Table 5.15: Coverage rates of the hospital dataset.

\mathfrak{s} (%)	5		10		50	
$(\mathfrak{w})\mathfrak{c}(\%)/\mathfrak{l}$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
<i>DSMC.Clo.Ind.c</i>	100.00	87.34	100.00	82.28	100.00	0
<i>DSMC.Clo.Ind.l</i>	100.00	87.34	100.00	79.75	100.00	0
<i>DSMC.Clo.Ind.wc</i>	100.00	89.87	100.00	84.81	100.00	0
<i>XRules.SR.Clo.Ind.c</i>	-	-	100.00	70.89	100.00	0
<i>XRules.SR.Clo.Ind.l</i>	-	-	100.00	81.01	100.00	0

in bold. The results at $\mathfrak{s}=50\%$, $(\mathfrak{w})\mathfrak{c}=90\%$ or $\mathfrak{l}=10$ are not shown as the accuracy and coverage rates are zeros.

It is seen that *XRules* gives better accuracy rates than *DSMC* except at $\mathfrak{s}=10\%$, $(\mathfrak{w})\mathfrak{c}=90\%$ or $\mathfrak{l}=10$, and $\mathfrak{s}=50\%$, $(\mathfrak{w})\mathfrak{c}=50\%$ or $\mathfrak{l}=1$. In the *DSMC* approach, the likelihood option often gives better accuracy rates than the other two options but it is not the case for *XRules*. In term of coverage rate, similar to the hospital dataset, both methods give the same results at default rule strength thresholds and the maximum coverage rates goes to the *DSMC* approach for remaining cases. The analysis of individual class results (not shown here) shows that *XRules* has better accuracy on all but at $\mathfrak{s}=10\%$ or $\mathfrak{s}=50\%$ for the minority class.

Table 5.16: Accuracy rates of the CRM dataset.

\mathfrak{s} (%)	1		5		10		50
$(\mathfrak{w})\mathfrak{c}(\%)/\mathfrak{l}$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)
<i>DSMC.Clo.Ind.c</i>	81.49	80.33	75.44	87.75	74.02	90.59	69.04
<i>DSMC.Clo.Ind.l</i>	81.14	85.07	78.29	87.79	75.80	92.54	72.60
<i>DSMC.Clo.Ind.wc</i>	78.65	80.09	77.58	87.36	75.44	91.67	72.60
<i>XRules.SR.Clo.Ind.c</i>	82.21	90.09	78.65	89.16	73.67	91.12	66.19
<i>XRules.SR.Clo.Ind.l</i>	77.94	84.00	79.36	88.44	76.87	92.03	72.40

Table 5.17: Coverage rates of the CRM dataset.

\mathfrak{s} (%)	1		5		10		50
$(\mathfrak{w})\mathfrak{c}(\%)/\mathfrak{l}$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)
<i>DSMC.Clo.Ind.c</i>	100.00	86.83	100.00	72.60	100.00	60.50	100.00
<i>DSMC.Clo.Ind.l</i>	100.00	78.65	100.00	61.21	100.00	47.69	100.00
<i>DSMC.Clo.Ind.wc</i>	100.00	80.43	100.00	64.77	100.00	51.25	100.00
<i>XRules.SR.Clo.Ind.c</i>	100.00	79.00	100.00	72.24	100.00	60.14	100.00
<i>XRules.SR.Clo.Ind.l</i>	100.00	80.07	100.00	61.57	100.00	49.11	99.29

5.5 Predicting Outcomes of Running Process Instances

In this section, the *DSMC* method is evaluated on the capability of predicting whether a process instance would complete successfully according to a predefined criteria. The experimental set up is that testing process instances only run through half way to their completion. The *DSMC* method is extended as a sequence of steps described in Table 5.18.

In this section, the experiments are performed on the hospital, the Dutch Financial Institute dataset (van Dongen, 2012), and the Repair Telephone dataset (used in Chapter 4) which will be presented in Section 5.5.1, 5.5.2, and 5.5.3, respectively.

5.5.1 Hospital Dataset

The hospital dataset was already mentioned in Section 3.3, 4.1.1, and 5.2. However, the dataset used here has two differences: (i) the whole dataset is used and (ii) the assignment of class labels to instances is based on their compliance to a predefined linear temporal logic rule. Specifically, the selected rule is $\phi = G(\text{“CEA – tumor marker using mea”} \rightarrow F(\text{“squamous cell carcinoma using eia”}))$ which is extracted from (Maggi et al., 2014). The parameter settings for the experiment are set as followed.

- Ten-fold cross validation is employed.
- Confidence thresholds are set at 50%, 60%, 70%, 80%, and 90%.
- Support thresholds are set at 1% and 5%.
- Root node is not removed.
- Closed and full tree are selected.
- 50 features are selected from the flat data (step 8 in Section 5.5).

The experimental results for support thresholds of 1% and 5% are displayed in Table 5.19 and 5.20, respectively. The same experiment was done in (Maggi et al., 2014). Note that in their study, ten-fold cross validation evaluation was not employed. Therefore their results might not be comparable with ours. Furthermore, in (Maggi et al., 2014) the training data are selected based on their similarity with the test instances, thus there might be biases occurring in the mining process.

The experimental results in the work of (Maggi et al., 2014) are displayed in Table 5.21. It can be seen that our method gives better *FI* and accuracy rates when confidence thresholds are set at 70% or higher for both support thresholds.

Table 5.18: A sequence of steps to predict an outcome of a running process instance.

<i>Step</i>	<i>Description</i>
1	Setting parameters: e.g. minimum supports, minimum confidences, subtree types, root node removal.
2	Preprocessing: attributes are sorted in the same order, <i>ID</i> attributes are removed, etc.
3	Assigning class labels to process instances based on desired business outcome, e.g. complete successfully/failed.
4	Converting the tree database into pre-order string encoding format.
5	Divided the dataset into ten different parts. The remaining steps are repeated ten times, each with one part used as test data and remaining parts as training data.
6	<i>DSM</i> extraction.
7	Converting training data to flat data format, e.g. <i>CSV</i> .
8	Extracting important feature: all values in the table are converted to nominal, <i>InfoGain</i> is used for the attribute rankings. The top <i>k</i> -features are selected for further processing whereas others are discarded.
9	Converting to itemset format and separate process instances to files based on class labels.
10	Removing event nodes to improve performance. However, this should not be done if induced subtrees are to be discovered.
11	Frequent itemset mining on each file—parameter can be normal, closed, or maximal patterns.
12	Merging all files containing the frequent itemset and from this file, reconstructing to either embedded, induced, or full tree.
13	Converting test data to flat data format using the <i>DSM</i> tree extracted from the training data.
14	Removing a half number of event nodes and their attributes in converted test instances to simulate running instances.
15	Converting training, test data and rules into itemset format.
16	Identifying rules' strength.
17	Identifying the total rule strength of matched rules for each test instance.
18	Classifying test instances based on the found total rule strengths.
19	Calculating performance rates such as <i>TPR</i> , <i>FPR</i> , <i>PPV</i> , <i>FI</i> , accuracy and coverage rate.

Table 5.19: Predicting whether running process instances comply to a linear temporal logic rule with a minimum support threshold of 1%–Hospital dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	87.4	3.8	21.0	1.8	98.0	84.7	80.6	88.5	80.0
$c = 60\%$	87.7	5.3	19.5	1.5	98.3	78.6	81.8	89.3	81.6
$c = 70\%$	86.8	11.3	13.5	2.4	97.3	54.4	86.5	91.6	86.1
$c = 80\%$	75.8	14.4	8.2	2.9	96.3	36.3	90.2	93.2	89.1
$c = 90\%$	71.8	14.9	5.4	2.8	96.2	26.6	93.0	94.6	91.3

Table 5.20: Predicting whether running process instances comply to a linear temporal logic rule with a minimum support threshold of 5%–Hospital dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	86.7	3.6	21.2	2.5	97.2	85.5	80.4	88.0	79.2
$c = 60\%$	87.8	4.1	20.7	1.4	98.4	83.5	80.9	88.8	80.6
$c = 70\%$	86.9	11.7	13.1	2.3	97.4	52.8	86.9	91.9	86.5
$c = 80\%$	67.8	13.7	4.2	2.5	96.4	23.5	94.2	95.3	92.7
$c = 90\%$	66.1	12.8	3.0	1.6	97.6	19.0	95.7	96.6	94.6

Table 5.21: Predicting whether running process instances comply to a linear temporal logic rule in (Maggi et al., 2014).

\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
110	19	7	35	94.0	35.1	85.2	89.4	84.7

The comparison of the two methods was also performed on predicting whether completed process instances comply to the selected linear temporal logic rule. The experimental settings remain the same. The *DSMC*'s results are shown in Table 5.22 and 5.23. The corresponding results in (Maggi et al., 2014) are displayed in Table 5.24. Similar to the case when process instances run half way through completion, *DSMC* gives better performance—in terms of *F1* and accuracy rates—when the confidence thresholds were increased to at least 70%.

Table 5.22: Predicting whether completed process instances comply to a linear temporal logic rule with a minimum support threshold of 1%–Hospital dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	87.7	4.8	20.0	1.5	98.3	80.6	81.4	89.1	81.1
$c = 60\%$	87.9	6.3	18.5	1.3	98.5	74.6	82.6	89.9	82.6
$c = 70\%$	86.8	12.2	12.6	2.4	97.3	50.8	87.3	92.0	86.8
$c = 80\%$	81.0	15.7	8.3	3.0	96.4	34.6	90.7	93.5	89.5
$c = 90\%$	77.9	16.6	6.0	3.1	96.2	26.5	92.8	94.5	91.2

Table 5.23: Predicting whether completed process instances comply to a linear temporal logic rule with a minimum support threshold of 5%–Hospital dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	86.7	4.4	20.4	2.5	97.2	82.3	81.0	88.4	79.9
$c = 60\%$	87.7	5.2	19.6	1.5	98.3	79.0	81.7	89.2	81.5
$c = 70\%$	87.1	13.0	11.8	2.1	97.6	47.6	88.1	92.6	87.8
$c = 80\%$	70.2	14.8	3.8	2.4	96.7	20.4	94.9	95.8	93.5
$c = 90\%$	68.9	14.3	2.8	1.6	97.7	16.4	96.1	96.9	95.1

Table 5.24: Predicting whether completed process instances comply to a linear temporal logic rule in (Maggi et al., 2014).

\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
315	50	25	103	92.6	32.6	86.3	89.3	84.7

5.5.2 Dutch Financial Institute Dataset

In this section, the *DFI* (van Dongen, 2012; Bautista et al., 2013) dataset is used to evaluate *DSMC*'s capability to predict outcomes of running process instances. The *DFI* data was originally in the *XES* format. Attributes such as registration date and events' timestamps are removed in the preprocessing step. Events having name as *A_PARTLYSUBMITTED*, *O_SELECTED*, *O_CREATED*, *O_ACCEPTED*, *A_REGISTERED*, or *A_ACTIVATED* are removed because they are redundant according to (Bautista et al., 2013). Additionally, the trace attribute *AMOUNT_REQ* is discretised into 10 bins. Finally, a process instance is assigned the class label *APPROVED* if its trace contains *A_APPROVED*; otherwise the class label *CANCELLEDORREJECTED* is assigned. The *XES* data is then converted to a rooted labelled ordered tree database which is represented in the pre-order string encoding. The structural properties of the converted data is described as follows: |transactions|= 13087; average encoding length (average number of nodes in pre-order encoding) = 141.8; maximum tree size = 661; average tree height = 2; average tree fan-out = 3.7; average tree size = 71.4; maximum tree height = 2; maximum tree fan-out = 167; average encoding length = 1321.

The parameter settings for the associative classification are set similarly to experiments done in Section 5.5.1 with the exception of support thresholds being set at 5% and 10%.

Table 5.25 and 5.26 show the results of the experiments when support thresholds were set at 5% and 10%, respectively. The *F1* and accuracy rates are promising, however when the confidence threshold reaches 90%, all performance indicators become less, exception for the *FPR* rate. It is noted that the coverage rate for the classification also drastically decreases.

Table 5.27 and 5.28 show the results of the experiments when support thresholds were set at 5% and 10%, respectively, and all test process instances complete. It can be seen in both settings that the number of true positives decreases, however, the number of true negative increases in greater number. As a result, the accuracy rate and *F1* values obtained for completed process instances are higher than those of running process instances. This is probably because it is harder predict correctly with lesser information.

Table 5.25: Predicting whether running process instances lead to a loan approval with a minimum support threshold of 5%–*DFI* dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	1071.8	32.8	191.7	11.7	98.9	85.4	84.8	91.3	84.4
$c = 60\%$	1070.6	36.9	187.6	12.9	98.8	83.6	85.1	91.4	84.7
$c = 70\%$	1068.6	46.0	178.5	14.9	98.6	79.5	85.7	91.7	85.2
$c = 80\%$	1072.6	33.4	191.1	10.9	99.0	85.1	84.9	91.4	84.6
$c = 90\%$	62.2	5.4	19.6	1.7	97.3	78.4	76.0	85.3	76.3

Table 5.26: Predicting whether running process instances lead to a loan approval with a minimum support threshold of 10%–*DFI* dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	1066.6	41.7	182.8	16.9	98.4	81.4	85.4	91.4	84.7
$c = 60\%$	1069.1	39.6	184.9	14.4	98.7	82.4	85.3	91.5	84.8
$c = 70\%$	1066.8	49.9	174.6	16.7	98.5	77.8	85.9	91.8	85.4
$c = 80\%$	1075.0	25.0	199.5	8.5	99.2	88.9	84.3	91.1	84.1
$c = 90\%$	48.2	0.3	12.7	0.2	99.6	97.7	79.1	88.2	79.2

Table 5.27: Predicting whether completed process instances get to a loan approval with a minimum support threshold of 5%–*DFI* dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	1030.5	155.4	69.1	53.0	95.1	30.8	93.7	94.4	90.7
$c = 60\%$	1029.5	155.6	68.9	54.0	95.0	30.7	93.7	94.3	90.6
$c = 70\%$	1025.6	154.7	69.8	57.9	94.7	31.1	93.6	94.1	90.2
$c = 80\%$	1040.4	139.6	84.9	43.1	96.0	37.8	92.5	94.2	90.2
$c = 90\%$	144.9	40.1	20.4	6.2	95.9	33.7	87.7	91.6	87.5

Table 5.28: Predicting whether completed process instances get to a loan approval with a minimum support threshold of 10%–*DFI* dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	1008.7	172.0	52.5	74.8	93.1	23.5	95.1	94.1	90.3
$c = 60\%$	1021.9	167.3	57.2	61.6	94.3	25.5	94.7	94.5	90.9
$c = 70\%$	1014.2	172.0	52.5	69.3	93.6	23.4	95.1	94.3	90.7
$c = 80\%$	1052.8	112.3	112.2	30.7	97.2	50.0	90.4	93.7	89.1
$c = 90\%$	114.3	2.0	12.8	0.6	99.5	86.5	89.9	94.5	89.7

5.5.3 Telephone Repair Dataset

In this section, the Telephone Repair dataset described in Section 4.3 is used to predict running process instances leading to a *simple* or *complex* case. The parameter settings for the associative classification are set similarly to experiments done in Section 5.5.1. However, the use of the feature selection step (step 8 in Section 5.5) in this dataset is not necessary.

The experimental results for predicting running process instances are presented in Table 5.29 and 5.30. It can be seen that the *F1* and accuracy rates are over 95% for most settings. Moreover, increasing the confidence thresholds improve the performance of the classifier. The experimental results for predicting running process instances are presented in Table 5.31 and 5.32. It is seen that the *F1* and accuracy rates are over 96% for all settings.

Table 5.29: Predicting running process instances would be labelled as simple or complex with a minimum support threshold of 1%–Telephone Repair dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	62.8	38.3	6.1	2.8	95.7	13.7	91.1	93.3	91.9
$c = 60\%$	62.6	42.1	2.3	2.9	95.6	5.2	96.5	96.0	95.3
$c = 70\%$	61.9	43.4	1.0	3.1	95.2	2.3	98.4	96.8	96.2
$c = 80\%$	60.1	43.6	0.3	3.2	94.9	0.7	99.5	97.1	96.7
$c = 90\%$	60.0	42.2	0.2	2.1	96.6	0.5	99.7	98.1	97.8

Table 5.30: Predicting running process instances would be labelled as simple or complex with a minimum support threshold of 5%–Telephone Repair dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	62.9	34.4	10.0	2.7	95.9	22.5	86.3	90.8	88.5
$c = 60\%$	62.6	42.0	2.4	2.9	95.6	5.4	96.3	95.9	95.2
$c = 70\%$	61.9	43.5	0.9	3.1	95.2	2.0	98.6	96.9	96.3
$c = 80\%$	60.0	43.6	0.0	3.2	94.9	0.0	100.0	97.4	97.0
$c = 90\%$	60.0	42.2	0.0	2.0	96.8	0.0	100.0	98.4	98.1

Table 5.31: Predicting completed process instances would be labelled as simple or complex with a minimum support threshold of 1%–Telephone Repair dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	62.6	44.3	0.1	3.0	95.4	0.2	99.8	97.6	97.2
$c = 60\%$	62.6	44.4	0.0	3.0	95.4	0.0	100.0	97.6	97.3
$c = 70\%$	62.9	44.4	0.0	2.7	95.9	0.0	100.0	97.9	97.5
$c = 80\%$	63.3	44.4	0.0	2.3	96.5	0.0	100.0	98.2	97.9
$c = 90\%$	64.4	44.4	0.0	1.2	98.2	0.0	100.0	99.1	98.9

Table 5.32: Predicting completed process instances would be labelled as simple or complex with a minimum support threshold of 5%–Telephone Repair dataset.

$c(\%)$	\overline{TP}	\overline{TN}	\overline{FP}	\overline{FN}	\overline{TPR}	\overline{FPR}	\overline{PPV}	$\overline{F1}$	\overline{ACC}
$c = 50\%$	61.7	44.4	0.0	3.9	94.1	0.0	100.0	97.0	96.5
$c = 60\%$	61.7	44.4	0.0	3.9	94.1	0.0	100.0	97.0	96.5
$c = 70\%$	61.7	44.4	0.0	3.9	94.1	0.0	100.0	97.0	96.5
$c = 80\%$	61.7	44.4	0.0	3.9	94.1	0.0	100.0	97.0	96.5
$c = 90\%$	61.8	44.4	0.0	3.7	94.4	0.0	100.0	97.1	96.6

5.5.4 Summary

The experimental results in Section 5.5.1, 5.5.2, and 5.5.3 show that the *DSMC* method is able to predict outcomes of running process instances. Therefore this method can be used to alert process owners when there is a high chance of unwanted outcomes.

The *DSMC* method is equipped with many parameters, e.g. subtree types, support thresholds, confidence thresholds, type of rule strength, etc. In our experiments, although only a few parameter settings were trialled, the results show high *F1* and accuracy rates. Note that in case the *DSMC* method demands exceptionally large computing power or storage space, more features should be removed from the data (step 8).

5.6 Recommendation Model for Running Process Instances

In this section, the *PCFSM* method is extended so that it is able to recommend subsequent actions for a running process instance in order to achieve a desired outcome. The sequence of steps performed in this experiment are described in Table 5.33.

The resulted classification model obtained after all steps in Table 5.33 are done could tell us which characteristics of future events are associated with a desired outcome.

Table 5.33: A sequence of steps to generate a recommendation model for a running process instance.

<i>Step</i>	<i>Description</i>
1	Setting parameters: e.g. minimum supports, minimum confidences, subtree types, root node removal.
2	Preprocessing: attributes are sorted in the same order, <i>ID</i> attributes are removed, etc.
3	Assigning class labels to process instances based on a desired business outcome, e.g. successfully completed.
4	Converting the tree database into pre-order string encoding format.
5	<i>DSM</i> extraction.
6	Converting training data to flat data format, e.g. <i>CSV</i> .
7	Converting test data to flat data format using the <i>DSM</i> tree extracted from the training data.
8	Selecting a test instance from the converted test data. In this experiment, the second half of the test instance's events are removed to simulate a running process instance.
9	Selecting instances in the training data that are similar (according to a pre-define threshold) to the test instance using a distance similarity measure, e.g. <i>Hamming</i> distance. Suppose that the test instance has n events, then only the first n events of the training instances are used for the similarity tests.
10	Forming a new training dataset based on the selected instances, however, the first n events for each instance are removed.
11	Applying a classification algorithm, e.g. the <i>C4.5</i> decision tree on the new training data.

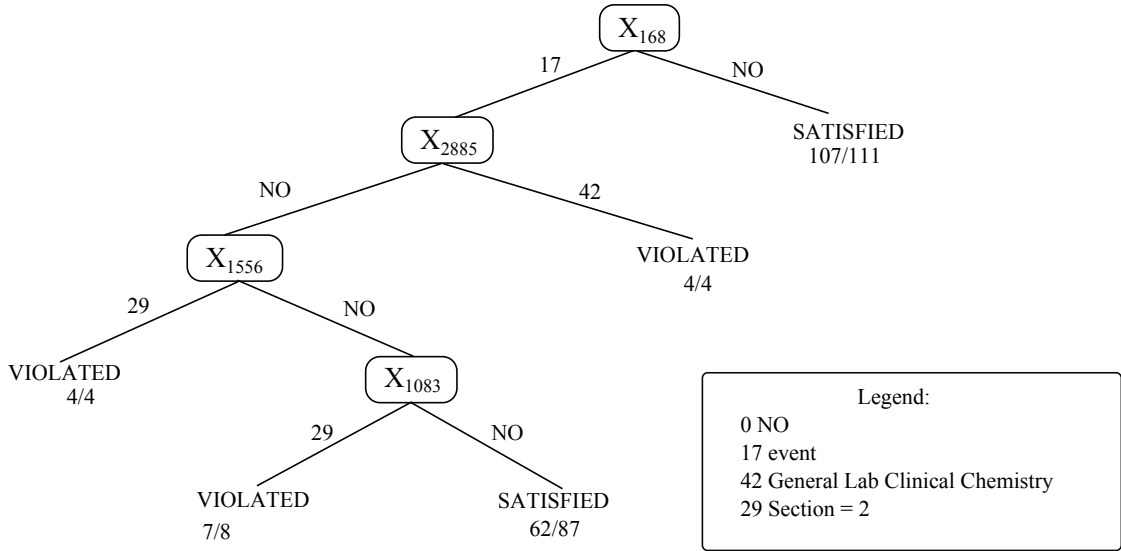


Figure 5.21: A (partial) recommendation model for a testing instance–Hospital dataset.

5.6.1 Recommendation Model for the Hospital Dataset

In this section, the hospital dataset described in Section 5.5.1 is examined. The parameter settings are described as follows.

- Support thresholds are set at 5%.
- Root node is removed.
- Distance threshold is set at 40%.

The selected test instance at the end of step 8 is “0 1 443 -1 -1 5 444 -1 -1 9 311 -1 -1 13 14 -1 -1 17 31 -1 14 -1 445 -1 120 -1 29 -1 446 -1 -1 17 31 -1 14 -1 121 -1 120 -1 29 -1 122 -1 -1 17 31 -1 14 -1 24 -1 120 -1 29 -1 25 -1 -1 17 42 -1 43 -1 44 -1 45 -1 46 -1 47 -1 -1”. At the end of step 9, 200 instances were formed for the new training dataset. Applying the *C4.5* decision tree algorithm to this dataset resulted in a recommendation model shown in Fig. 5.21. Note that only part of the model is shown because the classification rules obtained from the remaining leaves are insignificant (the ratio of correct predictions are small in those leaves).

To interpret the obtained recommendation model, the distance between event nodes should be identified. In the hospital dataset, this distance is 7. Therefore the running test process instance is currently at the 5th event. In Fig. 5.21, X_{168} , X_{1083} , X_{1556} and X_{2883} corresponds to the 22th, 152th, 221th, and 410th event of the process, respectively. From the above decision tree, it can be concluded that for the current running process to satisfy the linear temporal logic rule described in 5.5.1 with a confidence of 94% (107/111), this process should not have more than 21 events in total.

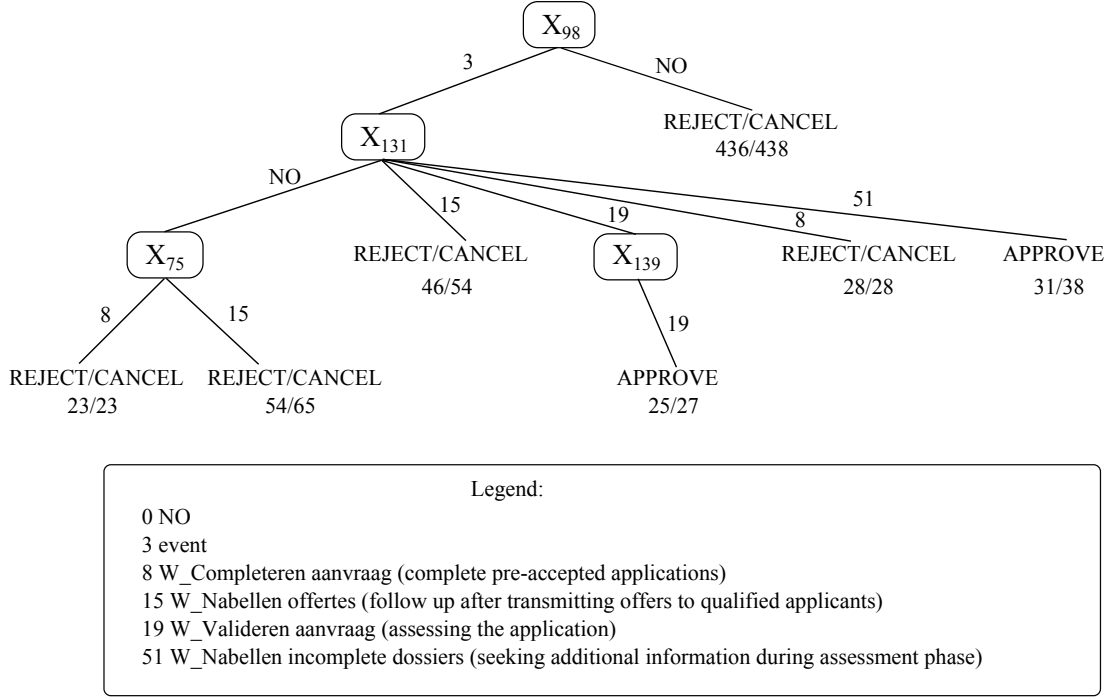


Figure 5.22: A (partial) recommendation model for a testing instance-*DFI* dataset.

5.6.2 Recommendation Model for the *DFI* Dataset

In this section, the *DFI* dataset described in Section 5.5.2 is examined. The parameter settings are described as follows.

- Support thresholds are set at 10%.
- Root node is removed.
- Distance threshold is set at 30%.

The selected test instance at the end of step 8 is “123 0 22 -1 3 4 -1 5 -1 6 -1 -1 3 41 -1 9 -1 6 -1 -1 3 41 -1 10 -1 62 -1 -1 3 41 -1 5 -1 62 -1 -1 3 41 -1 10 -1 76 -1 -1 3 41 -1 5 -1 76 -1 -1 3 41 -1 10 -1 52 -1 -1”. At the end of step 9, 179 instances were formed for the new training dataset. Applying the *C4.5* decision tree algorithm to this dataset resulted in a recommendation model shown in Fig. 5.22.

In the *DFI* dataset, the distance between event nodes is 5. As a result, X_{75} , X_{98} , X_{131} and X_{139} belong to the 15th, 20th, 26th, and 28th event of the process, respectively. It can be interpreted from the obtained recommendation model that for the current process instance to get a loan approval with a confidence of 92.6% (25/27), the 26th event should be *WNabellen incomplete dossiers* (Seeking additional information during assessment phase (Bautista et al., 2013)) and the 28th event should be *W_Valideren aanvraag* (Assessing the application).

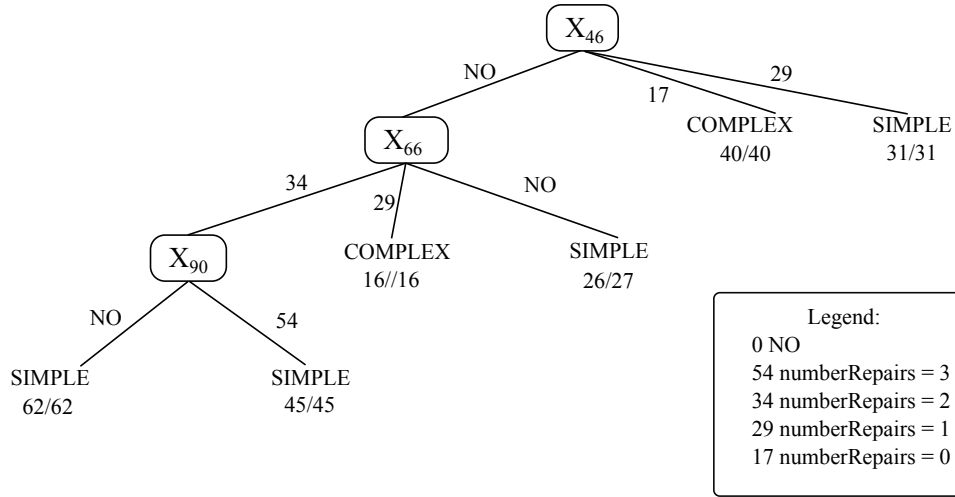


Figure 5.23: A (partial) recommendation model for a testing instance–Telephone Repair dataset.

5.6.3 Recommendation Model for the Telephone Repair Dataset

In this section, the Telephone Repair dataset described in Section 4.3 is examined. The parameter settings are described as follows.

- Support thresholds are set at 1%.
- Root node is removed.
- Distance threshold is set at 30%.

The selected test instance at the end of step 8 is “0 894 -1 3 -1 23 -1 5 6 -1 7 -1 8 -1 -1 5 43 -1 10 -1 11 -1 -1 5 43 -1 10 -1 8 -1 46 -1 13 -1 -1 5 27 -1 15 -1 11 -1 -1 5 27 -1 15 -1 8 -1 -1 5 9 -1 16 -1 11 -1 -1 5 9 -1 16 -1 8 -1 29 -1 30”. At the end of step 9, 250 instances were formed for the new training dataset. Applying the *C4.5* decision tree algorithm to this dataset resulted in a recommendation model shown in Fig. 5.23.

In the Telephone Repair dataset, the distance between event nodes varies because the number of attributes of each event can be different. From the flat data format of the training data, it can be found that X_{46} , X_{66} , and X_{90} belong to the 9th, 13th, and 18 event of the process, respectively. It can be interpreted from the obtained recommendation model that for the current process instance to be classified as a complex repair with a confidence of 100% (56/56), either the 9th event’s attribute should be “numberRepairs = 0” or the 13th event’s attribute should be “numberRepairs = 1”.

5.6.4 Summary

The *PCFSM* method was extended to build recommendation models for the Hospital, *DFI* and Telephone Repair datasets. The resulting recommendation models were easy to interpret and would serve as a reliable decision-making tool for process owners.

5.7 Enhancing *DSMC*

In this section, two methods of improving the *DSMC* method are introduced. First, a stable-rule-set method is used to reduce the number of rules. Second, a boosting method is applied to *DSMC* in order to improve coverage rates.

Stable-rule-set method Approaches that are based on the frequent pattern framework are often troubled by the sheer number of patterns discovered. A stable-rule-set method was developed for the purpose of obtaining a more compact rule set while maintaining accuracy and coverage rates.

The stable-rule-set method a four-step procedure which is described as follows.

1. The *DSM* is extracted from the training data.
2. The training data are randomly split into m partitions with equal number of instances.
3. Class-associative rules for each partition are formed using the method described in Section 5.1.2. The minimum support is adjusted using the following formula:

$$rs = rs * \frac{m-1}{m}.$$
4. The intersection of m class-associative rule sets forms the new rule set.

The accuracy and coverage results of the two methods on the *CSI-2* dataset are presented in Fig. 5.24 and 5.25, respectively. As can be seen from the figures, there are negligible differences among the results and the greatest difference is less than 1%, where the setting is $s = 2\%$, $c = 50\%$.

The classification accuracy of the two approaches on the hospital dataset is shown in Fig. 5.26. Except from the result where the stable set method has higher accuracy at $s = 5\%$, $c = 90\%$, the two methods' results are almost similar at other settings.

The results on coverage rates are not shown since they are identical between the two methods.

The accuracy and coverage rates of the two methods on the *CRM* dataset are shown in Fig. 5.27 and 5.28, respectively. The accuracy rates of the two methods are almost identical as the highest gap in accuracy rate is less than 1%, which occurs at $s = 10\%$, $c = 50\%$. The same characteristics can be found for the coverage rates, except that the stable-rule-set method is 13.27% less than the normal method at $s = 30\%$, $c = 90\%$.

Fig. 5.29, 5.30, and 5.31 show the number of rules extracted by the normal method and the stable-rule-set method. It can be seen that the number of rules obtained by the stable-rule-set method is consistently lower than that of the normal method.

Boosting

Ensemble methods such as bagging, boosting, and random forest tend to be effective in improving the classification accuracy (Han and Kamber, 2006). In this section,

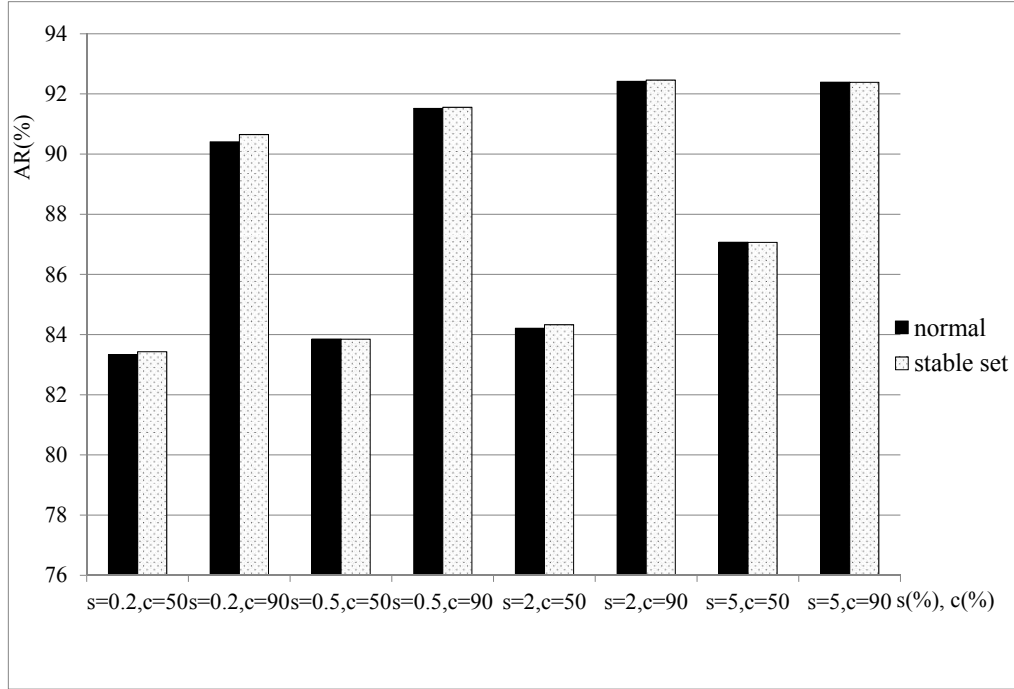


Figure 5.24: Accuracy rates of *DSMC* using the stable-rule-set and normal method on the *CSI-2* dataset.

ADABOOSTING (Freund and Schapire, 1997), which is a specific type of boosting methods, is applied to the *DSMC* method on the *CSI-2* dataset.

The accuracy and coverage rates obtained are displayed in Fig. 5.32 and 5.33. It is observed that the accuracy rates do not improve by using boosting method. However, the coverage rates improve 6% on average when using *ADABOOSTING*.

5.8 Conclusion

In this chapter, *DSMC*, an associative classifier that is based on the position-constrained subtree mining is proposed. This method is an extension of the *PCFSM* method described in Chapter 3. Hence, it is also applicable to process logs. The *DSMC* method was evaluated and compared with *XRULES*, a state-of-the-art tree-structured classifier, on several synthetic and real-world datasets.

The main characteristic of the synthetic datasets is that subtree patterns only occur (randomly) at pre-defined random locations in a random *DSM* tree. The experimental results of *DSMC* and *XRULES* on these datasets showed that the former method performs better in terms of accuracy rate. Both methods also have similar coverage rates; however the coverage rates of *DSMC* decrease more quickly than those of *XRULES* when minimum supports increase.

The *CSLOG* dataset represents for heterogeneous data. The experimental results

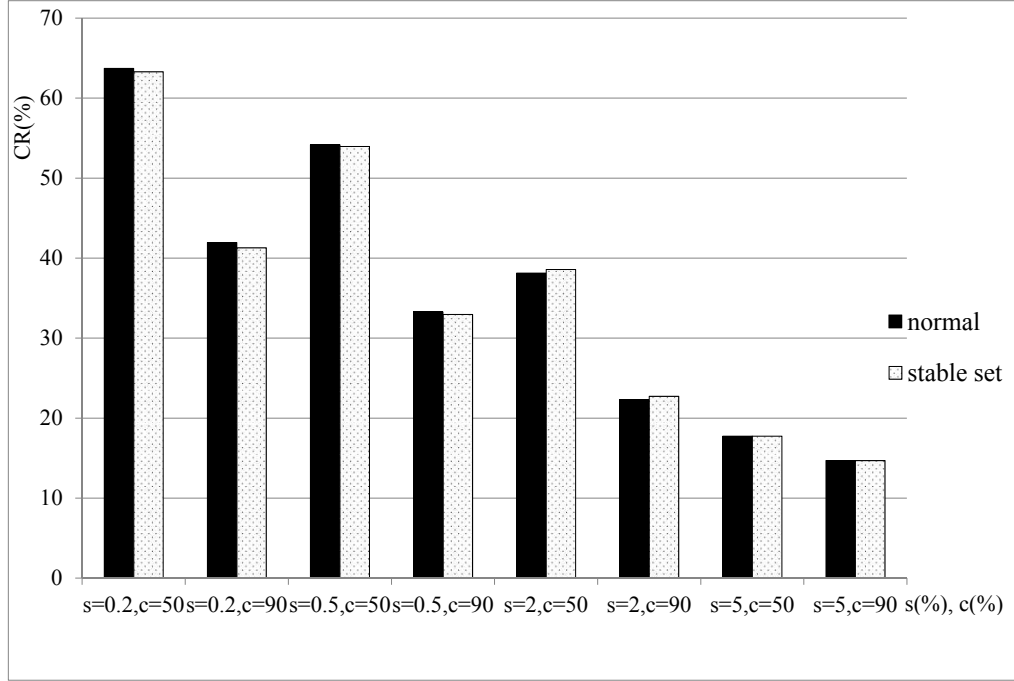


Figure 5.25: Coverage rates of *DSMC* using the stable-rule-set and normal method on the *CSI-2* dataset.

showed that the coverage rates of *DSMC* are less than those of *XRules*, which could be attributed to the limited number of patterns found when positional constraint being applied to frequent subtree mining. In terms of accuracy rate, the two methods are approximately the same when $s \leq 2\%$ and the configurations are *Frq.Emb.c* or *Clo.Ind.c*; for other configurations, the trends are not clear. In another evaluation setting, the student's t-test was used in conjunction with the ten-fold cross-validation method. The results showed that there are no statistical differences between the accuracy rates of the two approaches.

The hospital and the *CRM* datasets represent for homogeneous data. To enable *XRules* to work at lower support thresholds, all sibling nodes are renamed with distinct labels. The results on the modified datasets showed that *DSMC* has equal coverage rates to *XRules* at the default rule strength thresholds and higher maximum coverage rates for remained cases. In the hospital dataset, *DSMC* achieves better accuracy rates at $s < 10\%$ but *XRules* has higher accuracy rates for the remaining cases. In the *CRM* dataset, the accuracy rates of *XRules* are higher than those of *DSMC* when $s \leq 10\%$. In general, no clear trends in terms of accuracy rate have been found.

It is noticeable that *XRules* often produces larger rule sets, which possibly explains for its high coverage rates. Please note that due to the capability of *DSMC* to run at low support thresholds, the overall coverage rates of *DSMC* are comparable to those of *XRules*. For a fair comparison of accuracy rates, only subsets of test data where both classifiers cover are selected. It is remarkable that the resulting accuracy rates of the

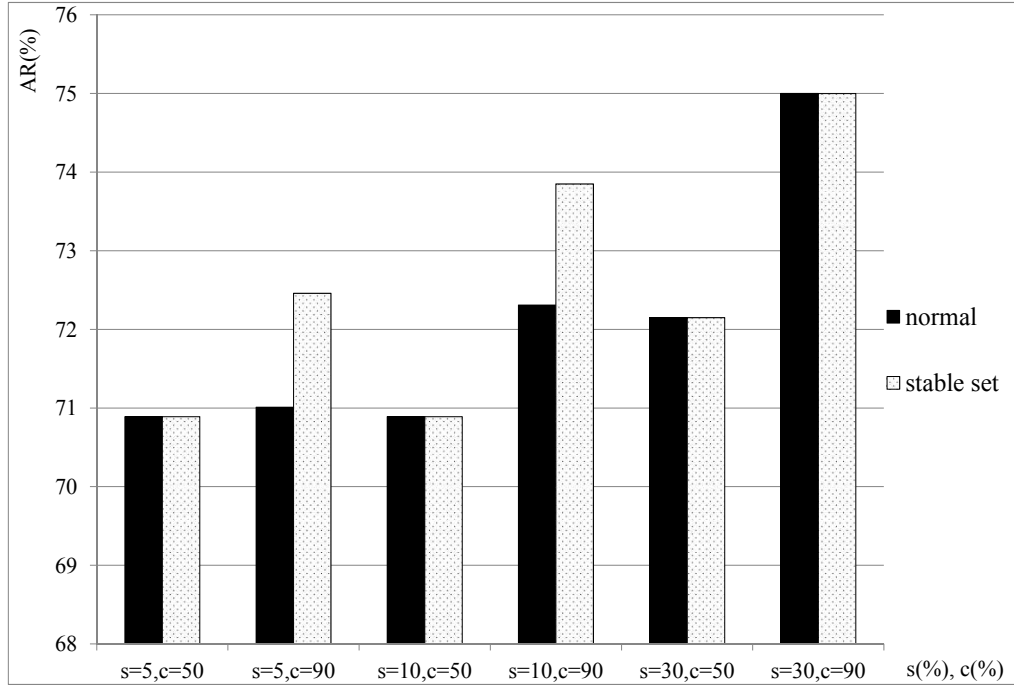


Figure 5.26: Accuracy rates of *DSMC* using the stable-rule-set and normal method on the hospital dataset.

two methods are comparable. In addition, for each test instance in these subsets of data, the number of *XRules*'s rules that are triggered is greater or equal to that of *DSMC*'s rules. This indicates the more focused nature of the position-constrained patterns. In a particular case, when *XRules*'s rules that have no counterparts in *DSMC* (rules that share the same subtrees) are removed, the accuracy rates of *XRules* are less than those of *DSMC*. This observation encourages for more studies to be conducted on finding effective strategies to combine the two approaches.

The evaluation of the two methods on majority and minority classes showed mixed results. It is worth noting that in both the hospital and *CRM* datasets, *XRules* with closed, induced subtree option failed to terminate after one day of running at any support threshold. For this reason, sibling nodes were renamed with distinct labels; however, by doing so the same strategy as *DSMC* is actually partially followed.

The evaluation was also performed using different options, such as *frequent*, *closed*, *induced*, *embedded* and *embedded-plus-disconnected* subtrees. Additionally, different rule strength measures, such as *confidence*, *weighted confidence* and *likelihood* were used as parameters for the comparisons. The closed, induced subtree options were preferred over frequent, embedded, or embedded-plus-disconnected subtree options because less patterns were generated (thus requires less space and decreases running time) while accuracy rates were not affected. The confidence measure was among the several rule strengths that consistently gave high accuracy and coverage rates. In addition, this measure is simple and can be easily explained to users. It was observed

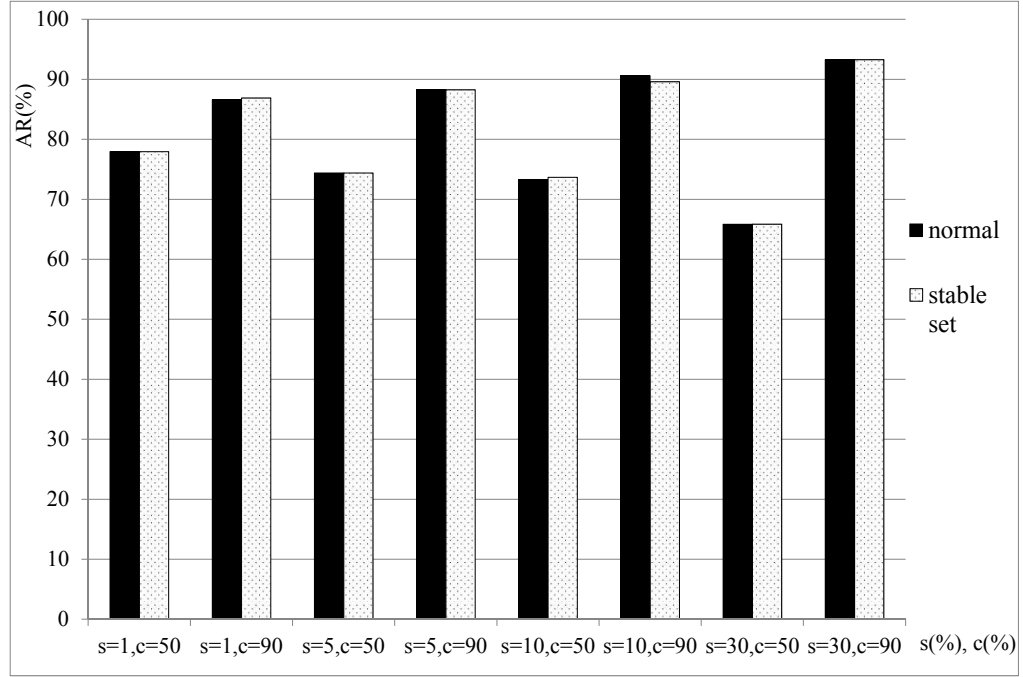


Figure 5.27: Accuracy rates of *DSMC* using the stable-rule-set and normal method on *CRM* dataset.

that using lower support thresholds often leads to better coverage rates, without affecting much on accuracy rates. On the other hand, using higher support thresholds may improve the accuracy rate but at the cost of a rapid decrease in the coverage rate.

Overall, the experimental results on homogeneous datasets showed that *DSMC* is a suitable classification method for tree-structured event data. *DSMC* works best when there is a need for identifying the exact location of subtree patterns. The method can also be used in complex datasets where other tree-structured classifiers struggle at high support thresholds. However, no claim is being made that this method can replace the classification methods that are based on traditional frequent subtree mining. In fact, they can be combined to produce a better classifier.

The *DSMC* method was extended for the purpose of predicting outcomes of running process instances. This functionality can be used as an alert system to notify process owners or managers when a running case has a high chance of being led to a failure or unwanted outcomes. Furthermore, this method was modified to be able to recommend suitable actions for a running process instance in order to achieve a desired outcome.

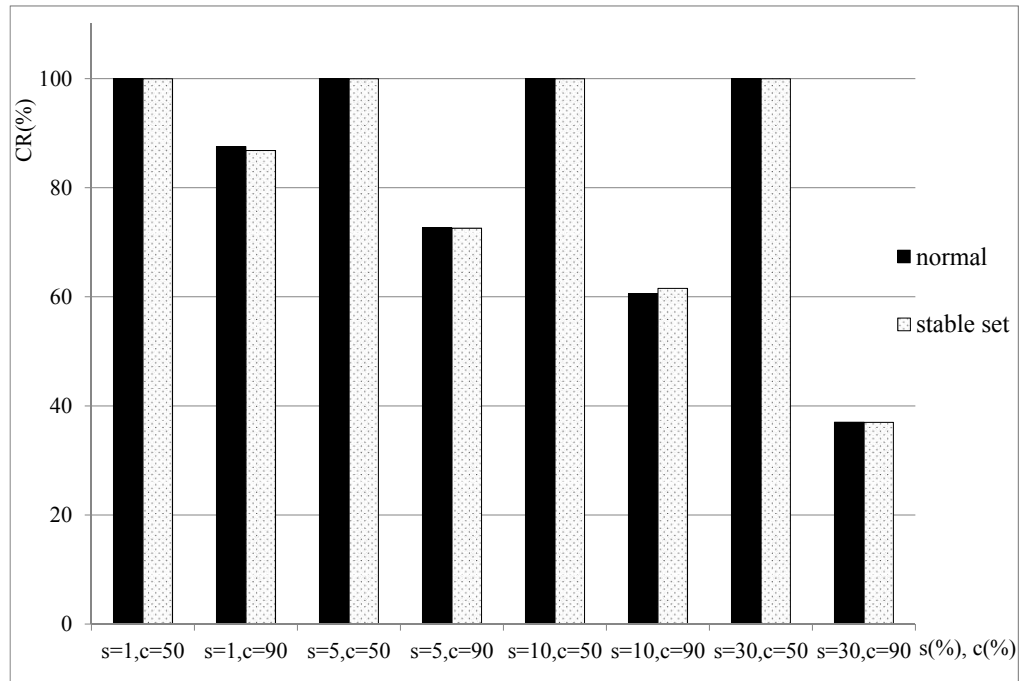


Figure 5.28: Coverage rates of *DSMC* using the stable-rule-set and normal method on *CRM* dataset.

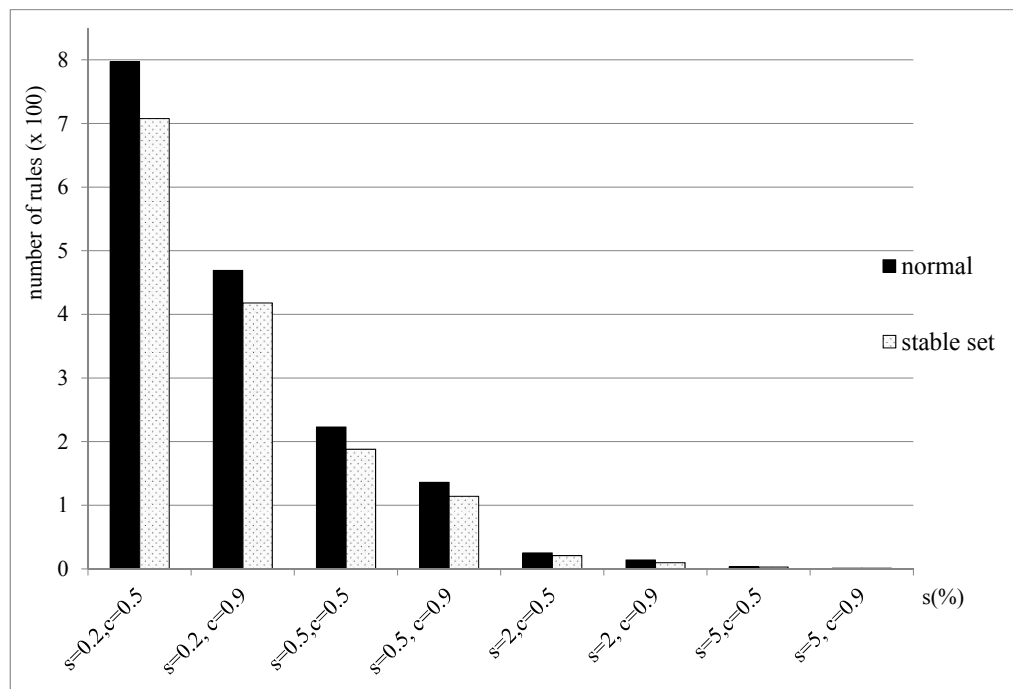


Figure 5.29: The number of rules of the stable-rule-set and normal methods on the *CSI-2* dataset.

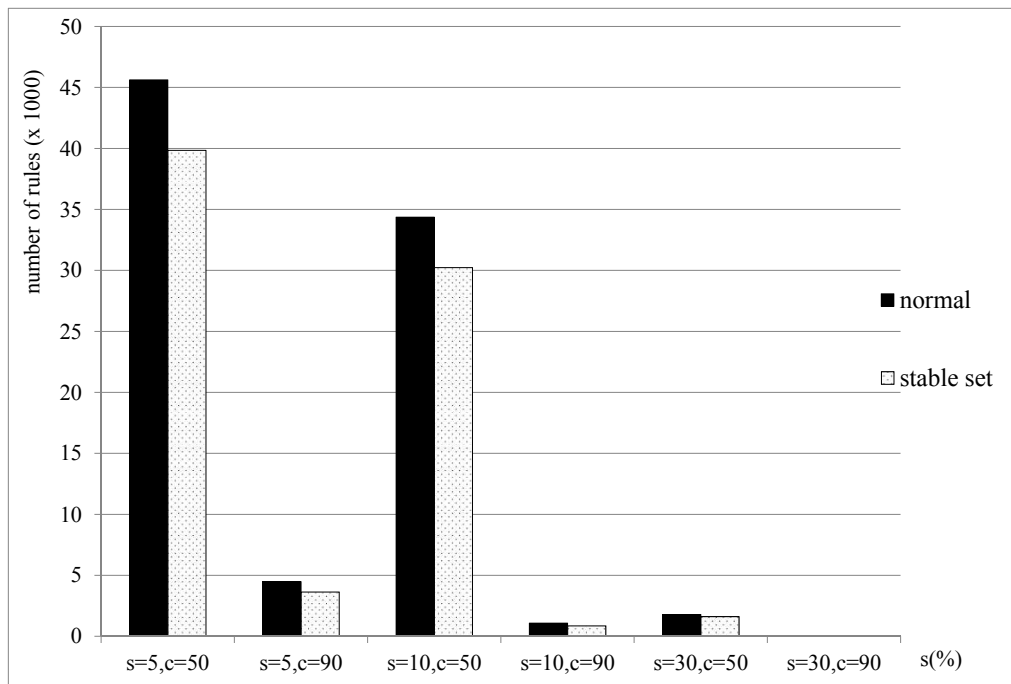


Figure 5.30: The number of rules of the stable-rule-set and normal methods on the hospital dataset.

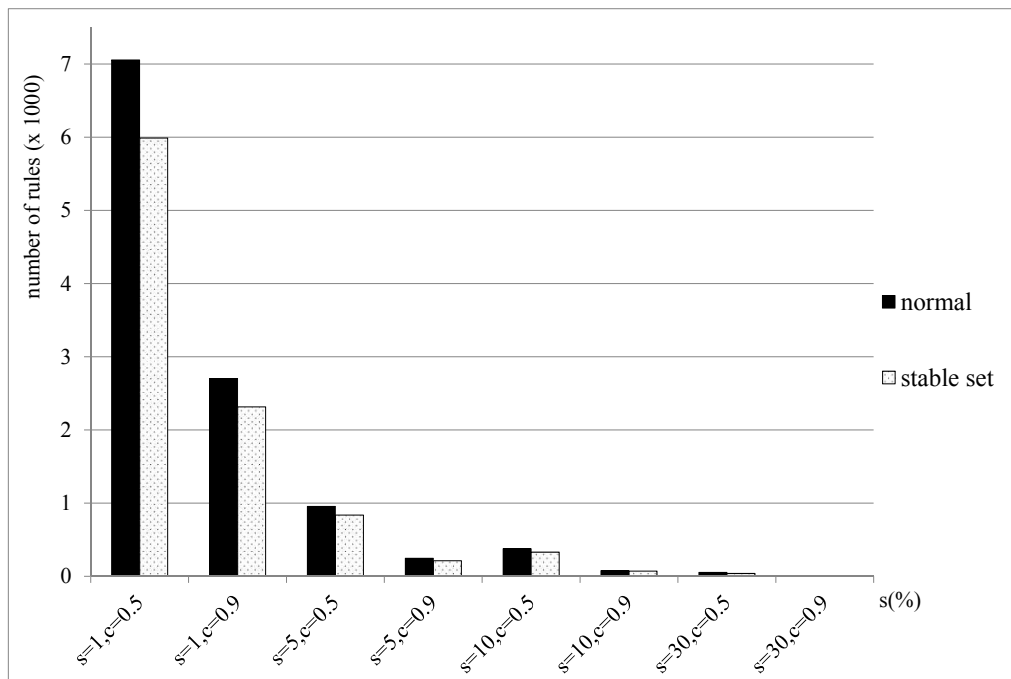


Figure 5.31: The number of rules of the stable-rule-set and normal methods on the CRM dataset.

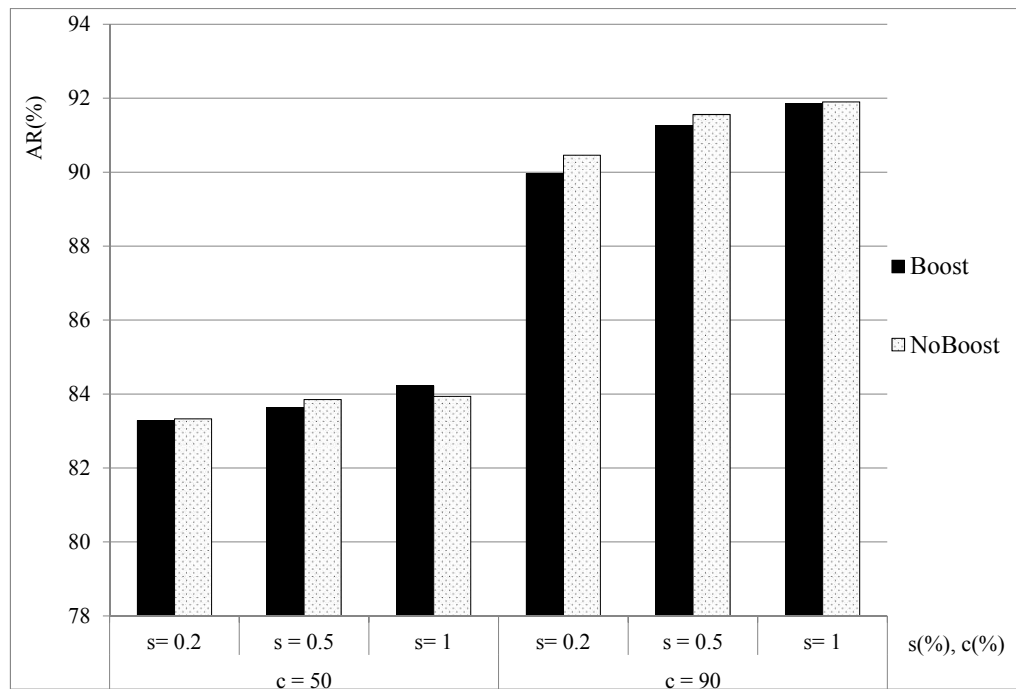


Figure 5.32: The effect of *ADABoosting* on the accuracy rates of *DSMC* on *CSI-2*.

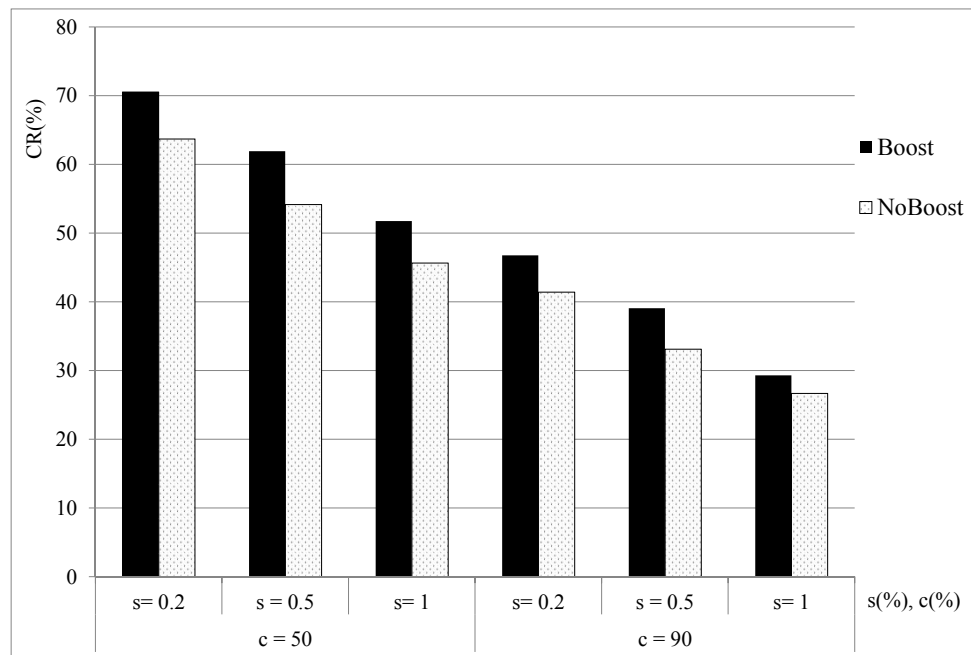


Figure 5.33: The effect of *ADABoosting* on the coverage rates of *DSMC* on *CSI-2*.

Chapter 6

Conclusion and Future Work

In this chapter, our main findings and outline future work are summarised. First, a recapitulation of the thesis is provided in Section 6.1. The contributions of this thesis are highlighted in Section 6.2; lessons learned from the results and implications of the study are also presented. Section 6.3 discusses the shortcomings of our work. Finally, in Section 6.4, several research directions that can be built upon our work are listed.

6.1 Recapitulation

Chapter 1 provided a background to business process management generally and process mining specifically. It is shown that the majority of studies in the process mining field is focused or based on process models, e.g. process discovery, process model enhancements, and conformance checking. Data mining algorithms were commonly used but often fulfilled a supplementary role. The question is asked as to whether a process log can be analysed without any knowledge of a process model. With the birth of *XML*-based standards for process logs and their increasing adoption, there has been a need to conduct a study on the use of *XML*/tree mining techniques to directly mine (or analyse) event data. To provide a clue as to how this can be done, several examples were given. The remainder of the chapter discussed the research questions, contributions and significance of the thesis.

Chapter 2 provided a comprehensive literature review and important notations used in the thesis. Firstly, the business process management field was introduced as a general context for our study. Secondly, the process mining field was described in detail with a focus on event logs and their data structures. Process model discovery algorithms, which were the main theme of research in the field, were also introduced to the readers. Methods that mine event logs without process models were further discussed. Thirdly, many data mining methods such as frequent pattern mining, classification, associative classification, as well as interestingness measures were presented. Fourthly,

the relationship between *XML* and tree-structured data were described and finally tree mining methods that are applicable to *XML*-based process logs are introduced.

Chapter 3 started with a motivating example in which the need for a tree-structured data mining applicable to process logs was highlighted. Next, some advantages of the position-constrained frequent subtree mining over traditional methods were discussed and a time performance study was conducted to evaluate these methods. The details of the structure-preserving tree mining approach and formal proofs were then presented. After that, an integrated method, called *PCFSM*, was proposed for the direct mining of process logs. This method was then extended with an exploratory analysis approach.

Chapter 4 reported the evaluation of the proposed methods. First, experimental settings and data descriptions were provided. Next, the exploratory analysis and the *PCFSM* method were evaluated. Then, a classification task, which is based on *PCFSM*, was tested on labelled process logs, highlighting the usefulness of the suggested method. The chapter ended with some lessons learned and general remarks.

Chapter 5 discussed an associative classification method for process logs. First, a literature review for *XML* classification was presented. Then, an associative classifier, named *DSMC*, was proposed based on the position-constrained frequent subtree mining. The method was evaluated in heterogeneous (*CSLOG* dataset) and homogeneous data (hospital and *CRM* dataset). The *DSMC* method was extended to predict outcomes of running process instances. In addition, it was modified to become an activity recommender for a running process to achieve a desired outcome. Lastly, several methods were proposed to improve *DSMC*'s classification accuracy and coverage rates, or to reduce the number of rules.

6.2 Contributions

The work in thesis has answered the research question proposed in Chapter 1 “In what way can *XML* data mining techniques be applied to *XML*-based event logs to discover hidden knowledge without using process models?”. An integrated method to analyse process log, called *PCFSM*, was proposed. A major component of this method is the structure-preserving flat data format for tree-structured data. The main idea is that by representing tree-structured process logs in a flat data format, a wide range of classical data mining methods such as classification, clustering, and outlier detection can be utilised.

The proposed approach is robust in terms of working under lower minimum supports as compared to other *FSM* or sequence mining methods. Being able to handle a lower minimum support threshold means that overall more patterns can be discovered, including the rare patterns which are potentially indicative of outlying/non-conforming process instances.

The *PCFSM* method was applied to real-world and synthetic process logs. The experimental results have shown the capability of the method in identifying the general characteristics of a group of process instances. In addition, by representing *XML*-based event logs in a flat data format, many classical data mining techniques can now be utilised for analysis purposes. As a case in point, a decision tree algorithm was used to predict an outcome of a process instance; e.g. to identify whether a claim would be rejected or accepted, or to identify performance bottlenecks, or e.g. to pick out process participants that are often associated with process delays.

Another contribution of the thesis is the *EPLA* method, which is particularly useful in the absence of domain knowledge to guide the analysis. It can also be used when the process analysts do not have a clear goal in mind, or they want to explore the data in the most generic way. To explore this process log in an unbiased manner, clustering techniques are used to detect groups of similar process executions, then *FSM* and classification techniques are used to identify the descriptive and distinguishing characteristics of the process instances, respectively. The suggested method is different from traditional clustering, in the sense that through the structure-preserved conversion of tree data, the structural characteristics of the process logs are taken into account. Furthermore, the position-constraint inherited through the approach will avoid grouping process instances that share similar events yet are completely different in regards to when and in which sequence each event took place.

A general purpose classification method for process logs, named *DSMC*, was proposed in this work. This method was built upon the associative classification framework and the position-constrained frequent/closed subtree mining approach. The classification accuracy of the proposed method is comparable to methods that are based on traditional frequent subtree mining. However, a particular characteristic of our proposed method which makes it suitable for process mining is that the subtrees and their nodes are labelled with absolute positions based on the *Document Structure Model*. Having each node in a tree instance being embedded with a position, it is possible to identify the context or location of frequent/outlying events and/or attributes. The *DSMC* classification model generally has fewer rules than other traditional methods for a given support, especially in datasets with more variations in structure. A smaller rule set would improve the time performance of an associative classifier and it would also be easier to interpret the rules; however, this may lead to less coverage. However, if the minimum support is reduced to a certain value in the *DSMC* method, different variants of the same rule in the traditional approach can be identified.

Since the schemas of *XES/MXML* are relatively simple and the number of events are often small, many repetitions—in terms of node label and structure—can be found in event data. For this reason, when applied to *XML*-based event logs, a traditional frequent subtree mining method often results in a large number of patterns, which

often lead to memory and storage problems. To alleviate the problem of having a large number of patterns when mining process logs, closed and/or induced subtree patterns are preferable to frequent/embedded subtree patterns. The experimental results showed that by using closed/induced subtree patterns, the number of rules was greatly reduced without affecting the classification accuracy or coverage rate.

Another strength of the proposed classification method is that different types of interestingness measure such as confidence, lift, and correlation can be used to filter out rules that have low discriminating power. The confidence measure is recommended for process analysts and domain experts due to its consistently high accuracy and coverage rates, and the definition of ‘confidence measure’ might be easier to understand for non-experts.

This thesis also explored the capability of the suggested methods in predicting whether a business process would lead to a successful completion. By extending the *DSMC* method, the method is now able to predict if a running process instance (have not completed) would eventually achieve a predefined outcome. Furthermore, the *PCFSM* method was extended to help process owners identify characteristics of future events of a running process instance that are associated with a successful completion (according to a desired outcome).

Overall, this work extends the available pool of process analysis techniques, allowing effective knowledge discovery from *XML*-based process logs in a more direct, unbiased manner and also provides an effective associative classifier for process log analysis. The process mining landscape is now broadened by the introduction of frequent subtree based techniques.

6.3 Limitations

For a given minimum support, the *DSMC*’s classification accuracy is comparable to that of other state-of-the-art methods on semi-structured event data; however, it does not cover as many instances as such. In practice, this is not a limitation as typically much lower support thresholds can be used to enumerate those additional patterns and potentially more subtree variants could be found (Hadzic et al., 2015). In cases where positional constraint on subtrees is not desired, traditional methods could be used. However, if there are complexity issues for the traditional approaches where low support thresholds cannot be handled, *DSMC* could be used with a small post-processing step to remove the position constraint and merge structure and label into one traditional subtree (Hadzic et al., 2015). On the other hand, in some cases covering more instances may not be desired, as the rules without position constraint could be considered too general (e.g. an association between two events without knowing where they have occurred within the process execution path may be unreliable for practical use).

Our proposed methods were evaluated mainly on one real-world and several synthetic process logs. We tried to obtain more real-world event logs, but due to the high confidentiality of this type of data, only one public dataset was used, i.e. the hospital dataset. Although the experiments confirm the validity of our methods, there is a need to further evaluate them on a wider range of datasets and process analysis applications.

6.4 Future Work

This section outlines several avenues for future exploration that were identified during the thesis work, but were outside of its scope.

6.4.1 Efficient User Interaction

In this thesis, the *PCFSM* method was used to identify the common characteristics of process instances or associations between event attributes. These tasks often result in a huge number of patterns; however, users are usually interested in specific properties of data/patterns. There should be a mechanism for more user interactions and control in the mining process so that unwanted results and running time are reduced. There have been many studies in pushing constraints into the sequential and association rule mining process and some notable works are (Ng et al., 1998), (Mooney and Roddick, 2013). Many constraints suggested in the above works are also applicable to our method, such as item, length, duration, and gap constraint. For example, a gap constraint for mining frequent characteristics of process logs could be that the time difference between two subsequent events must not be more than 1 day. Suppose that the event's time-stamps of this process log are located at positions $X_{(i+1)*8+3}$, $i \in \mathbb{N}$ in the *DSM*, e.g. X_3, X_{11}, X_{18} , the gap constraint can be formulated as $X_{(i+1)*8+3} - X_{i*8+3} \leq 1$. Basic constraints such as item constraint (patterns must contain a particular item) can be enforced in the pre-processing step; other types of constraint, such as gap constraint, have to be embedded in the mining algorithms.

Visualisation would help to convey the data and enable users to discover patterns effectively. It is important that the visualisation algorithm, developed for *PCFSM*, should be scalable to process large logs with millions of instances or those containing process instances with thousands of events. These situations are not uncommon in our current “Big Data” world. The tree-structured patterns should be able to link back to process instances where they occur. Data attributes that are not important or irrelevant, such as the *IDs*, could clutter the diagrams. Therefore, users may want to disable the appearance of these types of attribute.

It would be an important add-on to the proposed method if the *DSM* of a process log could be displayed in the form of an interactive map (van der Aalst, 2009). In this

map, users could zoom in and out on any specific part to gain more specific or general information, respectively. In our case, the zooming level could be associated with the current selected support thresholds. The *DSM* of a *XML*-based process, with nodes being annotated by their position, provides a general view of all process instances in the event log. When a user zooms in, the minimum support decreases and nodes with lower frequency are displayed. Using this technique, a user could observe the most important characteristics of an event log. This idea could be extended to displaying tree-structured patterns as well. In particular, when a specific part of a pattern is zoomed in (thereby decreasing the minimum support threshold), related patterns that are supersets of the nodes in view could also be displayed.

6.4.2 Combination of *DSMC* Rules and Traditional Frequent Subtree based Rules

From the experiments conducted in Chapter 5, it was seen that the accuracy rates of *DSMC* are similar to those of *XRules*, which is a traditional frequent subtree based associative classifier, despite its more compact rule set.

The work in (Hadzic et al., 2015) showed that position-constrained frequent subtree mining can operate at a much lower support threshold than traditional approaches. Additionally, more positioned-constrained variants of traditional frequent subtrees can be found at lower minimum supports. Hence, when it becomes infeasible for traditional methods to extract rules at lower support thresholds, *DSMC* can be used to add new rules to the rule set and more significant improvements in both accuracy and coverage rates are expected. In a preliminary experiment, the rule set of *XRules* was combined with the rule set of *DSMC* at a lower support, and the result was a better classifier that achieved better coverage rates than both methods. This initial result encourages the development of effective strategies to incorporate the two approaches.

6.4.3 Outlier Detection

Outlying process instances are those that deviate significantly from others. These instances could represent abnormal behaviours that require special attention, or frauds that should be prevented. The studies in (Bezerra et al., 2009) and (Depaire et al., 2013) first identify process models from event logs and process instances that do not conform to the models; these are classified as outliers. In (Folino et al., 2011), the authors introduce *S-patterns* and use such patterns for the clustering of event data based on the co-clustering method; instances that do not associate with any pattern cluster or belong to a cluster whose size is much smaller than other clusters' are considered outlying instances.

A possible avenue for future work is to utilise the *PCFSM* method for the clustering of tree-structured process logs. When the process instances are structurally varied, more clusters of smaller size are expected to appear, which could make it harder to distinguish outlying instances. One possible solution is to insert dummy nodes in the *DSM* and most common activities could be aligned to specific locations. By doing this, it would be easier to separate the outlying instances from the rest.

Appendix A

Experimental Results on CSLOG

Table A.1: Accuracy rates for the *CSI-2* dataset

$s(\%)$	0.2		0.5		1		2		5		10	
$(w)c(\%)/l$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
DSMC.Frq.Emb. s	83.24	90.38	83.87	91.57	83.93	91.86	84.22	92.42	87.07	92.39	92.39	92.39
DSMC.Clo.Emb. s	83.33	90.46	83.85	91.56	83.94	91.90	84.21	92.42	87.07	92.39	92.39	92.39
DSMC.Frq.Emb. l	80.40	87.59	82.10	87.79	81.92	86.23	81.81	0.00	78.59	0.00	78.98	0.00
DSMC.Clo.Emb. l	80.94	87.95	82.08	87.78	81.93	86.49	81.80	0.00	78.59	0.00	78.98	0.00
DSMC.Frq.Emb. wc	80.40	86.03	82.10	85.03	81.92	82.23	81.81	67.37	78.59	0.00	78.98	0.00
DSMC.Clo.Emb. wc	81.00	86.48	82.08	85.29	81.93	82.39	81.80	67.37	78.59	0.00	78.98	0.00
DSMC.Frq.Ind. s	83.24	90.38	83.87	91.57	83.93	91.86	84.22	92.42	87.07	92.39	92.39	92.39
DSMC.Clo.Ind. s	83.33	90.46	83.85	91.56	83.94	91.90	84.21	92.42	87.07	92.39	92.39	92.39
DSMC.Frq.Ind. l	80.50	87.61	82.10	87.79	81.92	86.23	81.81	0.00	78.59	0.00	78.98	0.00
DSMC.Clo.Ind. l	80.98	87.96	82.08	87.78	81.93	86.49	81.80	0.00	78.59	0.00	78.98	0.00
DSMC.Frq.Ind. wc	80.50	86.04	82.10	85.03	81.92	82.23	81.81	67.37	78.59	0.00	78.98	0.00
DSMC.Clo.Ind. wc	81.02	86.49	82.08	85.29	81.93	82.39	81.80	67.37	78.59	0.00	78.98	0.00
DSMC.Frq.Ftr. s	83.24	90.38	83.87	91.57	83.93	91.86	84.22	92.42	87.07	92.39	92.39	92.39
DSMC.Clo.Ftr. s	83.35	90.48	83.85	91.56	83.94	91.90	84.21	92.42	87.07	92.39	92.39	92.39
DSMC.Frq.Ftr. l	80.40	87.60	82.10	87.88	81.92	86.23	81.81	0.00	78.59	0.00	78.98	0.00
DSMC.Clo.Ftr. l	80.96	87.88	82.08	87.87	81.93	86.49	81.80	0.00	78.59	0.00	78.98	0.00
DSMC.Frq.Ftr. wc	80.40	86.00	82.10	85.03	81.92	82.23	81.81	67.37	78.59	0.00	78.98	0.00
DSMC.Clo.Ftr. wc	81.02	86.29	82.08	85.02	81.93	82.39	81.80	67.37	78.59	0.00	78.98	0.00
XRules.Frq.Emb. s	83.97	89.16	83.92	90.76	84.10	91.22	84.07	91.62	84.32	92.21	81.30	92.60
XRules.Frq.Emb. l	80.56	84.96	81.90	84.79	81.96	83.92	82.18	80.68	82.04	86.38	77.51	0.00
XRules.Clo.Ind. s	83.89	89.25	83.75	90.88	83.95	91.22	84.07	91.62	84.32	92.21	81.30	92.60
XRules.Clo.Ind. l	80.52	85.34	81.74	84.71	81.87	83.58	82.18	80.68	82.04	86.38	77.51	0.00

Table A.2: Accuracy rates for the CS2-3 dataset

$s(\%)$	0.2		0.5		1		2		5		10	
$(\mathfrak{w})c(\%)/l$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
DSMC.Frq.Emb. s	83.67	90.77	83.98	91.97	84.44	92.12	84.19	92.92	87.97	92.57	92.57	92.57
DSMC.Clo.Emb. s	83.82	90.97	83.99	92.01	84.45	92.16	84.19	92.92	87.97	92.57	92.57	92.57
DSMC.Frq.Emb. l	80.79	87.27	82.51	90.14	82.94	87.99	82.38	90.36	79.57	0.00	79.62	0.00
DSMC.Clo.Emb. l	81.18	87.78	82.56	90.22	82.95	87.96	82.38	90.24	79.57	0.00	79.62	0.00
DSMC.Frq.Emb. $\mathfrak{w}c$	80.83	87.14	82.51	89.60	82.94	87.99	82.38	90.36	79.57	0.00	79.62	0.00
DSMC.Clo.Emb. $\mathfrak{w}c$	81.16	87.58	82.56	89.67	82.95	87.96	82.38	90.24	79.57	0.00	79.62	0.00
DSMC.Frq.Ind. s	83.67	90.77	83.98	91.97	84.44	92.12	84.19	92.92	87.97	92.57	92.57	92.57
DSMC.Clo.Ind. s	83.82	90.97	83.99	92.01	84.45	92.16	84.19	92.92	87.97	92.57	92.57	92.57
DSMC.Frq.Ind. l	80.87	87.27	82.51	90.14	82.94	87.99	82.38	90.36	79.57	0.00	79.62	0.00
DSMC.Clo.Ind. l	81.18	87.78	82.56	90.22	82.95	87.96	82.38	90.24	79.57	0.00	79.62	0.00
DSMC.Frq.Ind. $\mathfrak{w}c$	80.89	87.14	82.51	89.60	82.94	87.99	82.38	90.36	79.57	0.00	79.62	0.00
DSMC.Clo.Ind. $\mathfrak{w}c$	81.16	87.58	82.56	89.67	82.95	87.96	82.38	90.24	79.57	0.00	79.62	0.00
DSMC.Frq.Ftr. s	83.67	90.77	83.98	91.97	84.44	92.12	84.19	92.92	87.97	92.57	92.57	92.57
DSMC.Clo.Ftr. s	83.82	90.91	83.99	92.01	84.45	92.16	84.19	92.92	87.97	92.57	92.57	92.57
DSMC.Frq.Ftr. l	80.79	87.27	82.51	90.10	82.94	89.14	82.38	90.36	79.57	0.00	79.62	0.00
DSMC.Clo.Ftr. l	81.18	87.73	82.56	90.16	82.95	89.12	82.38	90.24	79.57	0.00	79.62	0.00
DSMC.Frq.Ftr. $\mathfrak{w}c$	80.83	87.14	82.51	89.57	82.94	89.14	82.38	90.36	79.57	0.00	79.62	0.00
DSMC.Clo.Ftr. $\mathfrak{w}c$	81.16	87.58	82.56	89.62	82.95	89.12	82.38	90.24	79.57	0.00	79.62	0.00
XRules.Frq.Emb. s	84.59	89.63	84.87	91.11	84.26	91.49	84.26	92.64	85.00	93.55	82.00	92.83
XRules.Frq.Emb. l	79.92	82.98	82.14	87.05	82.58	88.26	82.60	89.51	82.78	84.40	78.35	0.00
XRules.Clo.Ind. s	84.64	89.94	84.79	91.12	84.26	91.49	84.26	92.64	85.00	93.55	82.00	92.83
XRules.Clo.Ind. l	81.17	84.77	82.08	87.31	82.58	88.11	82.60	89.51	82.78	84.40	78.35	0.00

Table A.3: Accuracy rates for the *CS3-I* dataset

$s(\%)$	0.2		0.5		1		2		5		10	
$(w)c(\%)/l$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
DSMC.Frq.Emb. s	83.96	90.86	84.71	92.04	83.20	92.16	83.63	92.68	80.08	92.38	80.17	92.38
DSMC.Clo.Emb. s	84.15	90.92	84.77	92.04	83.21	92.15	83.68	92.68	80.17	92.38	80.17	92.38
DSMC.Frq.Emb. l	80.75	87.47	82.33	88.50	82.90	88.60	83.61	92.46	80.08	0.00	80.17	0.00
DSMC.Clo.Emb. l	81.50	87.44	82.39	88.53	82.91	88.58	83.66	92.46	80.17	0.00	80.17	0.00
DSMC.Frq.Emb. wc	81.06	86.86	82.77	87.71	82.90	89.10	83.61	92.46	80.08	0.00	80.17	0.00
DSMC.Clo.Emb. wc	81.50	86.87	82.39	87.79	82.91	89.08	83.66	92.46	80.17	0.00	80.17	0.00
DSMC.Frq.Ind. s	84.00	90.86	84.71	92.04	83.20	92.16	83.63	92.68	80.08	92.38	80.17	92.38
DSMC.Clo.Ind. s	84.15	90.95	84.77	92.04	83.21	92.15	83.68	92.68	80.17	92.38	80.17	92.38
DSMC.Frq.Ind. l	80.95	87.47	82.33	88.50	82.90	88.60	83.61	92.46	80.08	0.00	80.17	0.00
DSMC.Clo.Ind. l	81.56	87.59	82.39	88.53	82.91	88.58	83.66	92.46	80.17	0.00	80.17	0.00
DSMC.Frq.Ind. wc	81.06	86.86	82.77	87.71	82.90	89.10	83.61	92.46	80.08	0.00	80.17	0.00
DSMC.Clo.Ind. wc	81.56	87.01	82.39	87.79	82.91	89.08	83.66	92.46	80.17	0.00	80.17	0.00
DSMC.Frq.Ftr. s	83.96	90.86	84.71	92.04	83.20	92.16	83.63	92.68	80.08	92.38	80.17	92.38
DSMC.Clo.Ftr. s	84.17	90.86	84.77	92.04	83.21	92.15	83.68	92.68	80.17	92.38	80.17	92.38
DSMC.Frq.Ftr. l	80.75	87.52	82.33	88.54	82.90	88.60	83.61	92.46	80.08	0.00	80.17	0.00
DSMC.Clo.Ftr. l	81.52	87.49	82.39	88.57	82.91	88.58	83.66	92.46	80.17	0.00	80.17	0.00
DSMC.Frq.Ftr. wc	81.06	86.91	82.77	87.73	82.90	89.10	83.61	92.46	80.08	0.00	80.17	0.00
DSMC.Clo.Ftr. wc	81.52	86.92	82.39	87.83	82.91	89.08	83.66	92.46	80.17	0.00	80.17	0.00
XRules.Frq.Emb. s	84.33	89.86	84.49	91.12	84.33	91.82	84.27	92.96	85.58	92.88	82.66	92.51
XRules.Frq.Emb. l	79.39	82.51	81.42	86.21	82.57	82.67	82.61	80.84	83.61	85.66	79.38	0.00
XRules.Clo.Ind. s	84.39	90.23	84.53	91.10	84.43	91.88	84.27	92.96	85.58	92.88	82.66	92.51
XRules.Clo.Ind. l	79.74	84.02	81.43	86.00	82.54	82.63	82.61	80.84	83.61	85.66	79.38	0.00

Table A.4: Coverage rates for the *CSI-2* dataset

$\mathfrak{s}(\%)$	0.2		0.5		1		2		5		10	
$(\mathfrak{w})\mathfrak{c}(\%)/\mathfrak{l}$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
DSMC.Frq.Emb. \mathfrak{s}	64.26	42.07	54.23	33.15	45.69	26.68	38.14	22.27	17.75	14.71	14.71	14.71
DSMC.Clo.Emb. \mathfrak{s}	63.71	41.72	54.16	33.12	45.65	26.67	38.12	22.27	17.75	14.71	14.71	14.71
DSMC.Frq.Emb. \mathfrak{l}	64.92	20.45	54.60	10.95	45.69	4.51	38.14	0.00	21.50	0.00	21.38	0.00
DSMC.Clo.Emb. \mathfrak{l}	64.22	20.04	54.53	10.93	45.65	4.49	38.12	0.00	21.50	0.00	21.38	0.00
DSMC.Frq.Emb. \mathfrak{wc}	64.92	23.09	54.60	12.98	45.69	6.91	38.14	1.28	21.50	0.00	21.38	0.00
DSMC.Clo.Emb. \mathfrak{wc}	64.22	22.57	54.53	12.85	45.65	6.90	38.12	1.28	21.50	0.00	21.38	0.00
DSMC.Frq.Ind. \mathfrak{s}	64.26	42.07	54.23	33.15	45.69	26.68	38.14	22.27	17.75	14.71	14.71	14.71
DSMC.Clo.Ind. \mathfrak{s}	63.71	41.72	54.16	33.12	45.65	26.67	38.12	22.27	17.75	14.71	14.71	14.71
DSMC.Frq.Ind. \mathfrak{l}	64.92	20.37	54.60	10.95	45.69	4.51	38.14	0.00	21.50	0.00	21.38	0.00
DSMC.Clo.Ind. \mathfrak{l}	64.22	19.96	54.53	10.93	45.65	4.49	38.12	0.00	21.50	0.00	21.38	0.00
DSMC.Frq.Ind. \mathfrak{wc}	64.92	23.01	54.60	12.98	45.69	6.91	38.14	1.28	21.50	0.00	21.38	0.00
DSMC.Clo.Ind. \mathfrak{wc}	64.22	22.49	54.53	12.85	45.65	6.90	38.12	1.28	21.50	0.00	21.38	0.00
DSMC.Frq.Ftr. \mathfrak{s}	64.26	42.08	54.23	33.15	45.69	26.68	38.14	22.27	17.75	14.71	14.71	14.71
DSMC.Clo.Ftr. \mathfrak{s}	63.79	41.81	54.16	33.12	45.65	26.67	38.12	22.27	17.75	14.71	14.71	14.71
DSMC.Frq.Ftr. \mathfrak{l}	64.92	20.68	54.60	11.03	45.69	4.51	38.14	0.00	21.50	0.00	21.38	0.00
DSMC.Clo.Ftr. \mathfrak{l}	64.30	20.27	54.53	11.01	45.65	4.49	38.12	0.00	21.50	0.00	21.38	0.00
DSMC.Frq.Ftr. \mathfrak{wc}	64.92	23.24	54.60	12.98	45.69	6.91	38.14	1.28	21.50	0.00	21.38	0.00
DSMC.Clo.Ftr. \mathfrak{wc}	64.30	22.84	54.53	12.97	45.65	6.90	38.12	1.28	21.50	0.00	21.38	0.00
XRules.Frq.Emb. \mathfrak{s}	82.10	58.77	73.61	49.51	64.95	39.96	54.39	32.23	42.08	24.96	23.82	15.32
XRules.Frq.Emb. \mathfrak{l}	82.26	31.14	73.73	21.20	64.95	9.57	54.39	6.36	42.08	3.17	23.82	0.00
XRules.Clo.Ind. \mathfrak{s}	81.28	57.90	73.34	48.67	64.17	39.21	54.38	32.23	42.08	24.96	23.82	15.32
XRules.Clo.Ind. \mathfrak{l}	81.40	32.58	73.46	20.04	64.17	9.21	54.38	6.36	42.08	3.17	23.82	0.00

Table A.5: Coverage rates for the CS2-3 dataset

$s(\%)$	0.2		0.5		1		2		5		10	
$(w)c(\%)/l$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
DSMC.Frq.Emb. s	64.05	36.08	54.02	28.74	46.41	24.61	38.40	19.99	17.66	14.81	14.81	14.81
DSMC.Clo.Emb. s	63.42	35.41	53.89	28.70	46.37	24.58	38.38	19.99	17.66	14.81	14.81	14.81
DSMC.Frq.Emb. l	64.62	21.72	54.26	12.77	46.41	5.35	38.40	1.09	21.37	0.00	21.36	0.00
DSMC.Clo.Emb. l	63.95	21.34	54.12	12.73	46.37	5.34	38.38	1.07	21.37	0.00	21.36	0.00
DSMC.Frq.Emb. wc	64.62	22.73	54.26	14.25	46.41	5.35	38.40	1.09	21.37	0.00	21.36	0.00
DSMC.Clo.Emb. wc	63.95	22.38	54.12	14.21	46.37	5.34	38.38	1.07	21.37	0.00	21.36	0.00
DSMC.Frq.Ind. s	64.05	36.08	54.02	28.74	46.41	24.61	38.40	19.99	17.66	14.81	14.81	14.81
DSMC.Clo.Ind. s	63.42	35.41	53.89	28.70	46.37	24.58	38.38	19.99	17.66	14.81	14.81	14.81
DSMC.Frq.Ind. l	64.62	21.72	54.26	12.77	46.41	5.35	38.40	1.09	21.37	0.00	21.36	0.00
DSMC.Clo.Ind. l	63.95	21.34	54.12	12.73	46.37	5.34	38.38	1.07	21.37	0.00	21.36	0.00
DSMC.Frq.Ind. wc	64.62	22.73	54.26	14.25	46.41	5.35	38.40	1.09	21.37	0.00	21.36	0.00
DSMC.Clo.Ind. wc	63.95	22.38	54.12	14.21	46.37	5.34	38.38	1.07	21.37	0.00	21.36	0.00
DSMC.Frq.Ftr. s	64.05	36.08	54.02	28.74	46.41	24.61	38.40	19.99	17.66	14.81	14.81	14.81
DSMC.Clo.Ftr. s	63.42	35.75	53.89	28.70	46.37	24.58	38.38	19.99	17.66	14.81	14.81	14.81
DSMC.Frq.Ftr. l	64.62	21.83	54.26	12.85	46.41	6.28	38.40	1.09	21.37	0.00	21.36	0.00
DSMC.Clo.Ftr. l	63.95	21.47	54.12	12.79	46.37	6.27	38.38	1.07	21.37	0.00	21.36	0.00
DSMC.Frq.Ftr. wc	64.62	22.84	54.26	14.33	46.41	6.28	38.40	1.09	21.37	0.00	21.36	0.00
DSMC.Clo.Ftr. wc	63.95	22.48	54.12	14.28	46.37	6.27	38.38	1.07	21.37	0.00	21.36	0.00
XRules.Frq.Emb. s	82.80	55.37	72.60	42.45	63.37	32.49	55.37	22.61	41.94	19.32	23.68	15.35
XRules.Frq.Emb. l	83.06	38.44	73.03	23.99	63.66	15.74	55.37	10.12	41.94	3.70	23.68	0.00
XRules.Clo.Ind. s	82.17	54.75	72.56	42.36	63.37	32.49	55.37	22.61	41.94	19.32	23.68	15.35
XRules.Clo.Ind. l	82.34	35.80	72.95	23.54	63.66	15.33	55.37	10.12	41.94	3.70	23.68	0.00

Table A.6: Coverage rates for the *CS3-I* dataset

$s(\%)$	0.2		0.5		1		2		5		10	
$(w)c(\%)/l$	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)	50(1)	90(10)
DSMC.Frq.Emb. s	64.09	35.92	53.94	29.09	46.67	24.80	37.07	19.63	20.46	14.79	20.42	14.79
DSMC.Clo.Emb. s	63.23	35.31	53.74	29.08	46.56	24.77	37.03	19.63	20.42	14.79	20.42	14.79
DSMC.Frq.Emb. l	64.68	22.13	55.29	12.50	47.08	5.76	37.09	3.12	20.46	0.00	20.42	0.00
DSMC.Clo.Emb. l	63.74	21.79	55.07	12.42	46.97	5.75	37.06	3.12	20.42	0.00	20.42	0.00
DSMC.Frq.Emb. wc	64.68	23.10	55.29	13.40	47.08	6.48	37.09	3.12	20.46	0.00	20.42	0.00
DSMC.Clo.Emb. wc	63.74	22.73	55.07	13.29	46.97	6.47	37.06	3.12	20.42	0.00	20.42	0.00
DSMC.Frq.Ind. s	64.09	35.92	53.94	29.09	46.67	24.80	37.07	19.63	20.46	14.79	20.42	14.79
DSMC.Clo.Ind. s	63.23	35.30	53.74	29.08	46.56	24.77	37.03	19.63	20.42	14.79	20.42	14.79
DSMC.Frq.Ind. l	64.68	22.13	55.29	12.50	47.08	5.76	37.09	3.12	20.46	0.00	20.42	0.00
DSMC.Clo.Ind. l	63.74	21.75	55.07	12.42	46.97	5.75	37.06	3.12	20.42	0.00	20.42	0.00
DSMC.Frq.Ind. wc	64.68	23.10	55.29	13.40	47.08	6.48	37.09	3.12	20.46	0.00	20.42	0.00
DSMC.Clo.Ind. wc	63.74	22.69	55.07	13.29	46.97	6.47	37.06	3.12	20.42	0.00	20.42	0.00
DSMC.Frq.Ftr. s	64.09	35.92	53.94	29.09	46.67	24.80	37.07	19.63	20.46	14.79	20.42	14.79
DSMC.Clo.Ftr. s	63.36	35.50	53.74	29.08	46.56	24.77	37.03	19.63	20.42	14.79	20.42	14.79
DSMC.Frq.Ftr. l	64.68	22.23	55.29	12.53	47.08	5.76	37.09	3.12	20.46	0.00	20.42	0.00
DSMC.Clo.Ftr. l	63.87	21.89	55.07	12.46	46.97	5.75	37.06	3.12	20.42	0.00	20.42	0.00
DSMC.Frq.Ftr. wc	64.68	23.19	55.29	13.43	47.08	6.48	37.09	3.12	20.46	0.00	20.42	0.00
DSMC.Clo.Ftr. wc	63.87	22.83	55.07	13.33	46.97	6.47	37.06	3.12	20.42	0.00	20.42	0.00
XRules.Frq.Emb. s	81.03	48.12	73.12	37.95	63.87	30.28	54.19	21.28	39.76	19.14	22.64	15.37
XRules.Frq.Emb. l	81.25	28.47	73.12	21.38	63.87	11.79	54.19	7.11	39.76	3.11	22.64	0.00
XRules.Clo.Ind. s	80.62	47.78	73.25	37.85	63.86	30.03	54.19	21.28	39.76	19.14	22.64	15.37
XRules.Clo.Ind. l	80.80	29.37	73.25	20.88	63.86	11.77	54.19	7.11	39.76	3.11	22.64	0.00

Index

DSM with a minimum support, 88

absolute support, 41

activity, 1

activity attribute, 30

as-is process model, 3

association rule mining, 46

associative classification, 53

AuditTrailEntry, 8

bootstrap, 52

BPM, 6

BPM life cycle, 2

C4.5, 48

candidate *DSM*, 76

case, 4, 29

Chi-square, 55

class accuracy, 121

class coverage, 121

class support, 119

closed patterns, 44

CLUTO, 104

CMTreeMiner, 61

combined rule strength, 120

confidence, 120

confusion matrix, 50

Conviction, 55

Cosine, 55

CRM, 130

cross-validation, 51

decision tree, 47

DryadeParent, 61

DSM tree, 77

DSMC, 122

embedded subtree, 58

event, 1, 4

event log, 1

external similarity, 104

forest, 58

frequent itemset mining, 42

frequent pattern, 41

frequent pattern mining, 40

frequent subsequence mining, 43

frequent subtree mining, 44, 61

GINI, 55

hospital dataset, 102

induced subtree, 58

inter-structure mining, 10

internal similarity, 104

intertestingness, 53

isomorphic subtree, 58

itemset, 42

Jaccard, 55

Laplace, 55

Leverage, 55

lift, 54

likelihood, 120

maximal patterns, 44

MXML, 6, 31

- overall accuracy, 121
- overall coverage, 121
- PAIS, 1
- PCFSM, 72
- Pearson, 55
- Petri net, 37
- pre-order string encoding, 60
- prediction, 6
- process analysis, 3
- process design, 3
- process enactment, 3
- process identification, 2
- process implementation and configuration, 3
- process instance, 4, 29
- process log, 1
- process mining, 4
- process monitoring and controlling, 3
- process redesign, 3
- recommendation, 7
- resource, 1
- root cause analysis, 5
- rooted ordered labelled, 57
- rooted ordered tree, 57
- rule accuracy, 49
- rule confidence, 46
- rule coverage, 49
- rule strength, 120
- rule support, 46
- sibling order, 72
- student's t-test, 52
- support, 41
- teleclaim dataset, 103
- telephone dataset, 103
- timestamp, 1, 29
- to-be process model, 3
- top-left mirror, 76
- top-left subtree, 72
- trace, 4, 30
- weighted confidence, 120
- weighted support, 59
- XES, 6, 31
- XRules, 93
- Yule's Q, 55

Bibliography

- Abiteboul, S. (1997). Querying semi-structured data. In Afrati, F. and Kolaitis, P., editors, *Proceedings of the 6th International Conference on Database Theory (ICDT'97)*, volume 1186, pages 1–18. Springer Berlin Heidelberg.
- Aggarwal, C. C. and Han, J. (2014). *Frequent Pattern Mining*. Springer International Publishing.
- Aggarwal, C. C., Ta, N., Wang, J., Feng, J., and Zaki, M. (2007). XProj: A framework for projected structural clustering of XML documents. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 46–55, New York, NY, USA. ACM.
- Agrawal, R., Gunopulos, D., and Leymann, F. (1998). Mining process models from workflow logs. In *Proceedings of the 6th International Conference on Extending Database Technology: Advances in Database Technology (EDBT'98)*, pages 469–483, London, UK. Springer-Verlag.
- Agrawal, R., Imielinski, T., and Swami, A. (1993). Mining association rules between sets of items in large databases. *Sigmod Record*, 22(2):207–216.
- Agrawal, R. and Srikant, R. (1994). Fast algorithms for mining association rules in large databases. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, pages 487–499, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Agrawal, R. and Srikant, R. (1995). Mining sequential patterns. In *Proceedings of the Eleventh International Conference on Data Engineering (ICDE'95)*, pages 3–14, Washington, DC, USA. IEEE Computer Society.
- Aguilar-Savén, R. S. (2004). Business process modelling: Review and framework. *International Journal of Production Economics*, 90(2):129–149.
- Aioli, F., Da San Martino, G., and Sperduti, A. (2009). Route kernels for trees. In *Proceedings of the 26th Annual International Conference on Machine Learning (ICML'09)*, pages 17–24, New York, NY, USA. ACM.

- Alberti, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P., and Torroni, P. (2008). Verifiable agent interaction in abductive logic programming: the SCIFF framework. *ACM Transactions on Computational Logic (TOCL)*, 9(4):29:1–29:43.
- Algergawy, A., Mesiti, M., Nayak, R., and Saake, G. (2011). XML data clustering: An overview. *ACM Computing Surveys*, 43(4):25:1–25:41.
- Arimura, H., Sakamoto, H., and Arikawa, S. (2001). Efficient learning of semi-structured data from queries. In *Proceedings of the 12th International Conference on Algorithmic Learning Theory*, pages 315–331. Springer.
- Arimura, H. and Uno, T. (2005). An output-polynomial time algorithm for mining frequent closed attribute trees. In Kramer, S. and Pfahringer, B., editors, *Inductive Logic Programming*, volume 3625 of *Lecture Notes in Computer Science*, pages 1–19. Springer Berlin Heidelberg.
- Asai, T., Abe, K., Kawasoe, S., Sakamoto, H., and Arikawa, S. (2002). Efficient sub-structure discovery from large semi-structured data. In *Proceedings of the Second SIAM International Conference on Data Mining (SIAM’02)*, pages 158–174.
- Asai, T., Arimura, H., Uno, T., and Nakano, S.-i. (2003). Discovering frequent sub-structures in large unordered trees. In Grieser, G., Tanaka, Y., and Yamamoto, A., editors, *Discovery Science*, volume 2843 of *Lecture Notes in Computer Science*, pages 47–61. Springer Berlin Heidelberg.
- Bautista, A., Wangikar, L., and Akbar, S. (2013). Process mining-driven optimization of a consumer loan approvals process. In La Rosa, M. and Soffer, P., editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 219–220. Springer Berlin Heidelberg.
- Beebe, N. H. F. (1993). Bibliography pretty printing and syntax checking. *TUGBoat*, 14(4):395–419.
- Bezerra, F., Wainer, J., and van der Aalst, W. M. P. (2009). Anomaly detection using process mining. In Halpin, T., Krogstie, J., Nurcan, S., Proper, E., Schmidt, R., Soffer, P., and Ukor, R., editors, *Enterprise, Business-Process and Information Systems Modeling*, volume 29 of *Lecture Notes in Business Information Processing*, pages 149–161. Springer Berlin Heidelberg.
- Blockeel, H. (1998). Top-down induction of first-order logical decision trees. *Artificial Intelligence*, 101(1-2):285–297.
- Blumberg, R. and Atre, S. (2003). The problem with unstructured data. *DM Review*, 13:42–49.

- Borgelt, C. and Berthold, M. R. (2002). Mining molecular fragments: Finding relevant substructures of molecules. In *Proceedings of the Second International Conference on Data Mining (ICDM'02)*, pages 211–218. IEEE.
- Bose, R. P. J. C. and Aalst, W. M. P. (2012). Analysis of patient treatment procedures. In Daniel, F., Barkaoui, K., Dustdar, S., Aalst, W., Mylopoulos, J., Rosemann, M., Shaw, M. J., and Szyperski, C., editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 165–166. Springer Berlin Heidelberg.
- Bozkaya, M., Gabriels, J., and van der Werf, J. M. (2009). Process diagnostics: a method based on process mining. In *Proceedings of the 2009 International Conference on Information, Process, and Knowledge Management (eKNOW'09)*, pages 22–27. IEEE.
- Bringmann, B. and Zimmermann, A. (2005). Tree²: Decision trees for tree structured data. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD'05)*, pages 46–58, Berlin, Heidelberg. Springer-Verlag.
- Bui, D. B., Hadzic, F., and Hecker, M. (2012a). Application of Tree-structured Data Mining for Analysis of Process Logs in XML format. In Zhao, Y., Li, J., Kennedy, P. J., and Christen, P., editors, *Proceedings of the 10th Australasian Data Mining Conference (AusDM 2012)*, pages 109–118, Sydney, Australia. ACS.
- Bui, D. B., Hadzic, F., and Hecker, M. (2013). Evaluation of Position-Constrained Association-Rule-Based Classification for Tree-Structured Data. In Li, J., Cao, L., Wang, C., Tan, K. C., Liu, B., Pei, J., and Tseng, V. S., editors, *Trends and Applications in Knowledge Discovery and Data Mining*, volume 7867 of *Lecture Notes in Computer Science*, pages 379–391. Springer Berlin Heidelberg.
- Bui, D. B., Hadzic, F., and Hecker, M. (2015). Direct and Unbiased Analysis of XML Process Logs Using Position and Structure Aware Data Mining. *Journal of Research and Practice in Information Technology*, 46(4). (to appear).
- Bui, D. B., Hadzic, F., and Potdar, V. (2012b). A Framework for Application of Tree-Structured Data Mining to Process Log Analysis. In Yin, H., Costa, J. A., and Barreto, G., editors, *Proceedings of the 13th International Conference on Intelligent Data Engineering and Automated Learning*, volume 7435 of *Lecture Notes in Computer Science*, pages 424–434. Springer.

- Bui, D. B., Hadzic, F., Tagarelli, A., and Hecker, M. (2014). Evaluation of an Associative Classifier based on Position-constrained Frequent/closed Subtree Mining. *Journal of Intelligent Information Systems*, 42(2):1–25.
- Candillier, L., Tellier, I., and Torre, F. (2006). Transforming XML trees for efficient classification and clustering. In *Proceedings of the 4th International Conference of the Initiative for the Evaluation of XML Retrieval (INEX'05)*, pages 469–480. Springer.
- Carmona, J., Cortadella, J., and Kishinevsky, M. (2009). Genet: A tool for the synthesis and mining of Petri nets. In *Proceedings of the Ninth International Conference on Application of Concurrency to System Design (ACSD'09)*, pages 181–185.
- Caron, F., Vanthienen, J., and Baesens, B. (2013). A comprehensive investigation of the applicability of process mining techniques for enterprise risk management. *Computer in Industry*, 64(4):464–475.
- Chehreghani, M. H., Chehreghani, M. H., Lucas, C., Rahgozar, M., and Ghadimi, E. (2009). Efficient rule based structural algorithms for classification of tree structured data. *Intelligent Data Analysis*, 13(1):165–188.
- Cheng, H., Yan, X., Han, J., and Yu, P. S. (2008). Direct discriminative pattern mining for effective classification. In *Proceedings of the 24th International Conference on Data Engineering*, pages 169–178. IEEE.
- Chi, Y., Muntz, R. R., Nijssen, S., and Kok, J. N. (2004a). Frequent subtree mining - an overview. *Fundamenta Informaticae*, 66(1-2):161–198.
- Chi, Y., Xia, Y., Yang, Y., and R. Muntz, R. (2005a). Mining closed and maximal frequent subtrees from databases of labeled rooted trees. *IEEE Transactions on Knowledge and Data Engineering*, 17(2):190–202.
- Chi, Y., Yang, Y., and Muntz, R. R. (2003). Indexing and mining free trees. In *Proceedings of the 3rd International Conference on Data Mining (ICDM'03)*, pages 509–512. IEEE.
- Chi, Y., Yang, Y., and Muntz, R. R. (2004b). HybridTreeMiner: An efficient algorithm for mining frequent rooted trees and free trees using canonical forms. In *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*, pages 11–20.
- Chi, Y., Yang, Y., and Muntz, R. R. (2005b). Canonical forms for labelled trees and their applications in frequent subtree mining. *Knowledge and Information Systems*, 8(2):203–234.

- Chi, Y., Yang, Y., Xia, Y., and Muntz, R. R. (2004c). CMTreeMiner: Mining both closed and maximal frequent subtrees. In Dai, H., Srikant, R., and Zhang, C., editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 63–73. Springer Berlin Heidelberg.
- Chowdhury, I. and Nayak, R. (2013). A novel method for finding similarities between unordered trees using matrix data model. In Lin, X., Manolopoulos, Y., Srivastava, D., and Huang, G., editors, *Web Information Systems Engineering WISE 2013*, volume 8180 of *Lecture Notes in Computer Science*, pages 421–430. Springer Berlin Heidelberg.
- Chowdhury, I. and Nayak, R. (2014). Boster: An efficient algorithm for mining frequent unordered induced subtrees. In Benatallah, B., Bestavros, A., Manolopoulos, Y., Vakali, A., and Zhang, Y., editors, *Web Information Systems Engineering WISE 2014*, volume 8786 of *Lecture Notes in Computer Science*, pages 146–155. Springer International Publishing.
- Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3(4):261–283.
- Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123. Morgan Kaufmann.
- Conforti, R., de Leoni, M., La Rosa, M., and van der Aalst, W. M. P. (2013). Supporting risk-informed decisions during business process execution. In Salinesi, C., Norrie, M. C., and Pastor, O., editors, *Advanced Information Systems Engineering*, volume 7908 of *Lecture Notes in Computer Science*, pages 116–132. Springer Berlin Heidelberg.
- Cook, J. E. and Wolf, A. L. (1995). Automating process discovery through event-data analysis. In *Proceedings of the 17th International Conference on Software Engineering (ICSE’95)*, pages 73–82, New York, NY, USA. ACM.
- Cook, J. E. and Wolf, A. L. (1998a). Discovering models of software processes from event-based data. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 7(3):215–249.
- Cook, J. E. and Wolf, A. L. (1998b). Event-based detection of concurrency. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering - SIGSOFT’98(FSE-6)*, pages 35–45, New York, NY, USA. ACM.
- Cook, J. E. and Wolf, A. L. (1999). Software process validation: Quantitatively measuring the correspondence of a process to a model. *ACM Transactions on Software Engineering and Methodology*, 8(2):147–176.

- Cooper, B., Sample, N., Franklin, M. J., Hjaltason, G. R., and Shadmon, M. (2001). A fast index for semistructured data. In *Proceedings of the 27th International Conference on Very Large Data Bases (VLDB'01)*, pages 341–350, San Francisco, CA, USA. Morgan Kaufmann.
- Cortadella, J., Kishinevsky, M., Lavagno, L., and Yakovlev, A. (1998). Deriving Petri nets from finite transition systems. *IEEE Transactions on Computers*, 47(8):859–882.
- Costa, G., Manco, G., Ortale, R., and Ritacco, E. (2013a). Hierarchical clustering of XML documents focused on structural components. *Data and Knowledge Engineering*, 84(0):26–46.
- Costa, G., Manco, G., Ortale, R., and Tagarelli, A. (2004). A tree-based approach to clustering XML documents by structure. In *Proceedings of the 8th European Conf. on Principles and Practice of Knowl. Discov. in Databases*, volume 3202 of *Lecture Notes in Computer Science*, pages 137–148.
- Costa, G., Ortale, R., and Ritacco, E. (2011). Effective XML classification using content and structural information via rule learning. In *Proceedings of the 23rd International Conference on Tools with Artificial Intelligence (ICTAI'11)*, pages 102–109, Washington, DC, USA. IEEE Computer Society.
- Costa, G., Ortale, R., and Ritacco, E. (2013b). X-Class: Associative classification of XML documents by structure. *ACM Transactions on Information Systems (TOIS)*, 31(1):3:1–3:40.
- Dalamagas, T., Cheng, T., Winkel, K.-J., and Sellis, T. (2005). Clustering XML documents using structural summaries. In Lindner, W., Mesiti, M., Trker, C., Tzitzikas, Y., and Vakali, A., editors, *Current Trends in Database Technology - EDBT 2004 Workshops*, volume 3268 of *Lecture Notes in Computer Science*, pages 547–556. Springer Berlin Heidelberg.
- Datta, A. (1998). Automating the discovery of as-is business process models: Probabilistic and algorithmic approaches. *Information Systems Research*, 9(3):275–301.
- Davenport, T. H. (1993). *Process Innovation: Reengineering Work Through Information Technology*. Harvard Business School Press, Boston, MA, USA.
- De Knijf, J. (2007). FAT-CAT: Frequent attributes tree based classification. In Fuhr, N., Lalmas, M., and Trotman, A., editors, *Comparative Evaluation of XML Information Retrieval Systems*, volume 4518 of *Lecture Notes in Computer Science*, pages 485–496. Springer Berlin Heidelberg.

- de Medeiros, A. K. A., van der Aalst, W. M. P., and Weijters, A. J. M. M. (2003). Workflow mining: Current status and future directions. In Meersman, R., Tari, Z., and Schmidt, D., editors, *On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE*, volume 2888 of *Lecture Notes in Computer Science*, pages 389–406. Springer Berlin Heidelberg.
- de Medeiros, A. K. A., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2007). Genetic process mining: an experimental evaluation. *Data Mining and Knowledge Discovery*, 14(2):245–304.
- De Weerd, J., Schupp, A., Vanderloock, A., and Baesens, B. (2013). Process mining for the multi-faceted analysis of business processes—a case study in a financial services organization. *Computers in Industry*, 64(1):57–67.
- Deepak, A., Fernández-Baca, D., Tirthapura, S., Sanderson, M. J., and McMahon, M. M. (2014). EvoMiner: Frequent subtree mining in phylogenetic databases. *Knowledge and Information Systems*, 41(3):559–590.
- Dehaspe, L., Toivonen, H., and King, R. D. (1998). Finding frequent substructures in chemical compounds. In Agrawal, R., Stolorz, P., and Piatetsky-Shapiro, G., editors, *Proceedings of the Fourth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD’98)*, pages 30–36. AAAI Press.
- Denoyer, L. and Gallinari, P. (2004). Bayesian network model for semi-structured document classification. *Information Processing and Management*, 40(5):807–827.
- Depaire, B., Swinnen, J., Jans, M., and Vanhoof, K. (2013). A process deviation analysis framework. In La Rosa, M. and Soffer, P., editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 701–706. Springer Berlin Heidelberg.
- Deshpande, M., Kuramochi, M., Wale, N., and Karypis, G. (2005). Frequent substructure-based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050.
- Dong, G., Zhang, X., Wong, L., and Li, J. (1999). CAEP: Classification by aggregating emerging patterns. In *Proceedings of the Second International Conference on Discovery Science (DS’99)*, pages 30–42. Springer.
- Dumas, M., La Rosa, M., Mendling, J., and Reijers, H. A. (2013). *Fundamentals of Business Process Management*. Springer, Berlin.

- Efron, B. and Tibshirani, R. (1997). Improvements on cross-validation: the 632+ bootstrap method. *Journal of the American Statistical Association*, 92(438):548–560.
- Egan, J. P. (1975). *Signal detection theory and ROC analysis*. Academic Press.
- Fayyad, U. M., Piatetsky-Shapiro, G., and Smyth, P. (1996). Knowledge discovery and data mining: Towards a unifying framework. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, pages 82–88.
- Feng, L. and Dillon, T. (2005). Mining interesting XML-enabled association rules with templates. In Goethals, B. and Siebes, A., editors, *Knowledge Discovery in Inductive Databases*, volume 3377 of *Lecture Notes in Computer Science*, pages 66–88. Springer Berlin Heidelberg.
- Feng, L., Dillon, T., Weigand, H., and Chang, E. (2003). An XML-enabled association rule framework. In *Proceedings the 14th International Conference on Database and Expert Systems Applications*, volume 2736 of *Lecture Notes in Computer Science*, pages 88–97. Springer Berlin Heidelberg.
- Ferreira, D. R. and Gillblad, D. (2009). Discovering process models from unlabelled event logs. In *Proceedings of the 7th International Conference on Business Process Management (BPM'09)*, pages 143–158. Springer-Verlag.
- Ferreira, H. M. and Ferreira, D. R. (2006). An integrated life cycle for workflow management based on learning and planning. *International Journal of Cooperative Information Systems*, 15(04):485–505.
- Folino, F., Greco, G., Guzzo, A., and Pontieri, L. (2011). Mining usage scenarios in business processes: Outlier-aware discovery and run-time prediction. *Data and Knowledge Engineering*, 70(12):1005–1029.
- Freund, Y. and Schapire, R. E. (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139.
- Friedrich, F., Mendling, J., and Puhlmann, F. (2011). Process model generation from natural language text. In Mouratidis, H. and Rolland, C., editors, *Advanced Information Systems Engineering*, volume 6741 of *Lecture Notes in Computer Science*, pages 482–496. Springer Berlin Heidelberg.
- Garboni, C., Masseglia, F., and Trousse, B. (2006). Sequential pattern mining for structure-based XML document classification. In *Proceedings of the 4th Interna-*

- tional Conference on Initiative for the Evaluation of XML Retrieval (INEX'05)*, pages 458–468. Springer-Verlag.
- Geamsakul, W., Yoshida, T., Ohara, K., Motoda, H., Yokoi, H., and Takabayashi, K. (2005). Constructing a decision tree for graph-structured data and its applications. *Fundamenta Informaticae*, 66(1):131–160.
- Geng, L. and Hamilton, H. J. (2006). Interestingness measures for data mining: A survey. *ACM Computing Surveys (CSUR)*, 38(3).
- Ghionna, L., Greco, G., Guzzo, A., and Pontieri, L. (2008). Outlier detection techniques for process mining applications. In An, A., Matwin, S., Ras, Z. W., and Slezak, D., editors, *Foundations of Intelligent Systems*, volume 4994 of *Lecture Notes in Computer Science*, pages 150–159. Springer Berlin Heidelberg.
- Goedertier, S., De Weerd, J., Martens, D., Vanthienen, J., and Baesens, B. (2011). Process discovery in event logs: An application in the telecom industry. *Applied Soft Computing*, 11(2):1697–1710.
- Goedertier, S., Martens, D., Vanthienen, J., and Baesens, B. (2009). Robust process discovery with artificial negative events. *The Journal of Machine Learning Research*, 10:1305–1340.
- Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10(5):447–474.
- Goldfarb, C. F. and Rubinsky, Y. (1990). *The SGML Handbook*. Oxford University Press.
- Gonçalves, M. A., Luo, M., Shen, R., Ali, M. F., and Fox, E. A. (2002). An XML log standard and tool for digital library logging analysis. In *Proceedings of the 6th European Conference on Research and Advanced Technology for Digital Libraries*, pages 129–143.
- Greco, G., Guzzo, A., Manco, G., and Sacca, D. (2005). Mining and reasoning on workflows. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):519–534.
- Greco, G., Guzzo, A., Pontieri, L., and Sacca, D. (2006). Discovering expressive process models by clustering log traces. *IEEE Transactions on Knowledge and Data Engineering*, 18(8):1010–1027.
- Grigori, D., Casati, F., Castellanos, M., Dayal, U., Sayal, M., and Shan, M.-C. (2004). Business process intelligence. *Computers in Industry*, 53(3):321–343.

- Gross, D. and Harris, C. (1998). *Fundamentals of queueing theory*. Wiley Interscience.
- Günther, C. W. and van Der Aalst, W. M. P. (2007). Fuzzy mining: Adaptive process simplification based on multi-perspective metrics. In Alonso, G., Dadam, P., and Rosemann, M., editors, *Proceedings of the International Conference on Business Process Management - BPM '07*, Lecture Notes in Computer Science, pages 328–343. Springer, Berlin.
- Hadzic, F. (2012). A structure preserving flat data format representation for tree-structured data. In Cao, L., Huang, J. Z., Bailey, J., Koh, Y., and Luo, J., editors, *New Frontiers in Applied Data Mining*, volume 7104 of *Lecture Notes in Computer Science*, pages 221–233. Springer Berlin Heidelberg, Shenzhen, China.
- Hadzic, F., Hecker, M., and Tagarelli, A. (2011a). XML document clustering using structure-preserving flat representation of XML content and structure. In *Proceedings of the 7th International Conference on Advanced Data Mining and Applications (ADMA'11)*, pages 403–416. Springer-Verlag.
- Hadzic, F., Hecker, M., and Tagarelli, A. (2015). Ordered subtree mining via transactional mapping using a structure-preserving tree database schema. *Information Sciences*. (to appear).
- Hadzic, F., Tan, H., and Dillon, T. S. (2007). UNI3-efficient algorithm for mining unordered induced subtrees using TMG candidate generation. In *Proceedings of the IEEE Symposium on Computational Intelligence and Data Mining (CIDM 2007)*, pages 568–575. IEEE.
- Hadzic, F., Tan, H., and Dillon, T. S. (2011b). *Mining of Data with Complex Structures*, volume 333. Springer.
- Han, J., Cheng, H., Xin, D., and Yan, X. (2007a). Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86.
- Han, J., Cheng, H., Xin, D., and Yan, X. (2007b). Frequent pattern mining: Current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86.
- Han, J. and Kamber, M. (2006). *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 3 edition.
- Han, J., Pei, J., and Yin, Y. (2000). Mining frequent patterns without candidate generation. *ACM SIGMOD Record*, 29(2):1–12.

Han, J., Wang, J., Lu, Y., and Tzvetkov, P. (2002). Mining top-k frequent closed patterns without minimum support. In *Proceedings of the IEEE International Conference on Data Mining (ICDM'02)*, pages 211–218, Washington, DC, USA. IEEE Computer Society.

Hashimoto, K., Takigawa, I., Shiga, M., Kanehisa, M., and Mamitsuka, H. (2008). Mining significant tree patterns in carbohydrate sugar chains. *Bioinformatics*, 24(16):i167–i173.

Herbst, J. (2000). A machine learning approach to workflow management. In *Proceedings of the 11th European Conference on Machine Learning*, Lecture Notes in Computer Science, pages 183–194. Springer-Verlag.

Herbst, J. and Karagiannis, D. (1998). Integrating learning and workflow management to support acquisition and adaptation of workflow models. In *Proceedings of the Ninth International Workshop on Database and Expert Systems Applications*, pages 745–752. IEEE.

Herbst, J. and Karagiannis, D. (2004). Workflow mining with InWoLvE. *Computers in Industry*, 53(3):245–264.

Hido, S. and Kawano, H. (2005). AMIOT: Induced ordered tree mining in tree-structured databases. In *Proceedings of the Fifth IEEE International Conference on Data Mining (ICDM'05)*, pages 170–177. IEEE.

Hopcroft, J. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley.

Hovy, E., Navigli, R., and Ponzetto, S. P. (2013). Collaboratively built semi-structured content and artificial intelligence: The story so far. *Artificial Intelligence*, 194:2–27.

Huan, J., Wang, W., and Prins, J. (2003). Efficient mining of frequent subgraphs in the presence of isomorphism. In *Proceedings of the Third International Conference on Data Mining (ICDM'03)*, pages 549–552. IEEE.

Hwang, S.-Y., Wei, C.-P., and Yang, W.-S. (2004). Discovery of temporal patterns from process instances. *Computers in Industry*, 53(3):345–364.

ICTFPM (2010). XES Working Group. http://www.processmining.org/event_logs_and_models_used.in
(This dataset is accompanied with the book (van der Aalst, 2011a). Last accessed 19 May 2015).

Jain, A. K., Murty, M. N., and Flynn, P. J. (1999). Data clustering: A review. *ACM Computing Survey*, 31(3):264–323.

- Jans, M., van der Werf, J. M., Lybaert, N., and Vanhoof, K. (2011). A business process mining application for internal transaction fraud mitigation. *Expert Systems with Applications*, 38(10):13351–13359.
- Jiang, C., Coenen, F., and Zito, M. (2013). A survey of frequent subgraph mining algorithms. *The Knowledge Engineering Review*, 28(01):75–105.
- Jiménez, A., Berzal, F., and Cubero, J.-C. (2010). POTMiner: Mining ordered, unordered, and partially-ordered trees. *Knowledge and Information Systems*, 23(2):199–224.
- Kaplan, R. S. and Norton, D. P. (2005). The balanced scorecard: Measures that drive performance. *Harvard Business Review*, 83(7):172–180.
- Karuppusami, G. and Gandhinathan, R. (2006). Pareto analysis of critical success factors of total quality management: A literature review and analysis. *The TQM magazine*, 18(4):372–385.
- Karypis, G. (2003). CLUTO: A clustering toolkit. Technical report, University of Minnesota.
- Kashima, H. and Koyanagi, T. (2002). Kernels for semi-structured data. In *Proceedings of the Nineteenth International Conference on Machine Learning (ICML'02)*, pages 291–298, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.
- Kim, H., Kim, S., Weninger, T., Han, J., and Abdelzaher, T. (2010). NDPMine: Efficiently mining discriminative numerical features for pattern-based classification. In *Proceedings of the European Conference on Machine learning and Knowledge Discovery in Databases: Part II (ECML PKDD'10)*, pages 35–50. Springer-Verlag.
- Kim, K. (2006). A XML-based workflow event logging mechanism for workflow mining. In *Proceedings of the 2006 International Conference on Advanced Web and Network Technology and Applications (APWeb'06)*, pages 132–136.
- Kuramochi, M. and Karypis, G. (2001). Frequent subgraph discovery. In *Proceedings of the 2001 International Conference on Data Mining (ICDM'01)*, pages 313–320. IEEE.
- Kurgan, L. A. and Musilek, P. (2006). A survey of knowledge discovery and data mining process models. *Knowledge Engineering Review*, 21(1):1–24.
- Kutty, S., Nayak, R., and Li, Y. (2011). XML documents clustering using a tensor space model. In *Proceedings of the 15th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining - Part I (PAKDD'11)*, pages 488–499, Berlin, Heidelberg. Springer-Verlag.

- Lamma, E., Mello, P., Montali, M., Riguzzi, F., and Storari, S. (2007). Inducing declarative logic-based models from labeled traces. In Alonso, G., Dadam, P., and Rosemann, M., editors, *Business Process Management*, volume 4714 of *Lecture Notes in Computer Science*, pages 344–359. Springer Berlin Heidelberg.
- Lavrac, N. and Dzeroski, S. (1993). *Inductive Logic Programming: Techniques and Applications*. Routledge, New York, NY.
- Li, W., Han, J., and Pei, J. (2001). CMAR: Accurate and efficient classification based on multiple class-association rules. In *Proceedings of the 2001 International Conference on Data Mining (ICDM'01)*, pages 369–376. IEEE.
- Linstone, H. A. and Turoff, M. (1975). *The Delphi Method: Techniques and Applications*, volume 29. Addison-Wesley.
- Liu, B., Hsu, W., and Ma, Y. (1998). Integrating classification and association rule mining. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 80–86. AAAI Press.
- Madria, S. K., Bhowmick, S. S., Ng, W. K., and Lim, E. P. (1999). Research Issues in Web Data Mining. In *Proceedings of the First International Conference on Data Warehousing and Knowledge Discovery (DaWak'99)*, pages 303–312. Springer.
- Maggi, F. M., Di Francescomarino, C., Dumas, M., and Ghidini, C. (2014). Predictive monitoring of business processes. In Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., and Horkoff, J., editors, *Advanced Information Systems Engineering*, volume 8484 of *Lecture Notes in Computer Science*, pages 457–472. Springer International Publishing.
- Makanju, A., Zincir-Heywood, A. N., and Milios, E. E. (2012). A lightweight algorithm for message type extraction in system application logs. *IEEE Transactions on Knowledge and Data Engineering*, 24(11):1921–1936.
- Mannila, H. and Rusakov, D. (2001). Decomposition of event sequences into independent components. In *Proceedings of the First SIAM International Conference on Data Mining (SDM'01)*, pages 1–17. SIAM.
- Mans, R. S., Schonenberg, M. H., Song, M., van der Aalst, W. M. P., and Bakker, P. J. M. (2009). Application of process mining in healthcare a case study in a Dutch hospital. In Fred, A., Filipe, J., and Gamboa, H., editors, *Biomedical Engineering Systems and Technologies*, volume 25 of *Communications in Computer and Information Science*, pages 425–438. Springer Berlin Heidelberg.

- Mărușter, L., Weijters, A. J. M. M. T., Van Der Aalst, W. M. P., and Van Den Bosch, A. (2006). A rule-based approach for process discovery: Dealing with noise and imbalance in process logs. *Data Mining and Knowledge Discovery*, 13(1):67–87.
- Mazuran, M., Quintarelli, E., and Tanca, L. (2009a). Mining tree-based frequent patterns from XML. In Andreasen, T., Yager, R. R., Bulskov, H., Christiansen, H., and Larsen, H., editors, *Flexible Query Answering Systems*, volume 5822 of *Lecture Notes in Computer Science*, pages 287–299. Springer Berlin Heidelberg.
- Mazuran, M., Quintarelli, E., and Tanca, L. (2009b). Mining tree-based frequent patterns from XML. In Andreasen, T., Yager, R. R., Bulskov, H., Christiansen, H., and Larsen, H., editors, *Flexible Query Answering Systems*, volume 5822 of *Lecture Notes in Computer Science*, pages 287–299. Springer Berlin Heidelberg.
- Mierswa, I., Wurst, M., Klinkenberg, R., Scholz, M., and Euler, T. (2006). YALE: Rapid prototyping for complex data mining tasks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, pages 935–940, New York, NY, USA. ACM.
- Miyahara, T., Shoudai, T., Uchida, T., Takahashi, K., and Ueda, H. (2001). Discovery of frequent tree structured patterns in semistructured web documents. In Cheung, D., Williams, G. J., and Li, Q., editors, *Advances in Knowledge Discovery and Data Mining*, volume 2035 of *Lecture Notes in Computer Science*, pages 47–52. Springer Berlin Heidelberg.
- Mooney, C. H. and Roddick, J. F. (2013). Sequential pattern mining – approaches and algorithms. *ACM Computing Surveys*, 45(2):19:1–19:39.
- Moradi, M. and Keyvanpour, M. R. (2015). An analytical review of XML association rules mining. *Artificial Intelligence Review*, 43(2):277–300.
- Muehlen, M. Z. and Rosemann, M. (2000). Workflow-based process monitoring and controlling-technical and organizational issues. In *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, pages 1–10, Los Almitos, CA, USA. IEEE Computer Society Press.
- Nakatumba, J., Westergaard, M., and van der Aalst, W. M. P. (2012). An infrastructure for cost-effective testing of operational support algorithms based on colored Petri nets. In Haddad, S. and Pomello, L., editors, *Application and Theory of Petri Nets*, volume 7347 of *Lecture Notes in Computer Science*, pages 308–327. Springer Berlin Heidelberg.
- Nayak, R. (2008). Fast and effective clustering of XML data using structural information. *Knowledge and Information Systems*, 14(2):197–215.

- Nayak, R., Witt, R., and Tonev, A. (2002). Data mining and XML documents. In *Proceedings of the International Conference on Internet Computing, IC'02*, Las Vegas, Nevada, USA.
- Ng, R. T., Lakshmanan, L. V. S., Han, J., and Pang, A. (1998). Exploratory mining and pruning optimizations of constrained associations rules. *SIGMOD Record*, 27(2):13–24.
- Nigam, A. and Caswell, N. S. (2003). Business artifacts: An approach to operational specification. *IBM Systems Journal*, 42(3):428–445.
- Nijssen, S. and Kok, J. N. (2003). Efficient discovery of frequent unordered trees. In *Proceedings of the First International Workshop on Mining Graphs, Trees and Sequences (MGTS'03)*, pages 55–64.
- Nijssen, S. and Kok, J. N. (2004). A quickstart in frequent structure mining can make a difference. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'04)*, pages 647–652, New York, NY, USA. ACM.
- Ordys, A., Uduehi, D., and Johnson, M. A. (2007). *Process Control Performance Assessment: From Theory to Implementation*. Springer, London.
- Pasquier, N., Bastide, Y., Taouil, R., and Lakhal, L. (1999). Discovering frequent closed itemsets for association rules. In *Proceedings of the 7th International Conference on Database Theory (ICDT'99)*, pages 398–416, London, UK. Springer-Verlag.
- Pei, J., Han, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., and Hsu, M.-C. (2001). Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 29th International Conference on Data Engineering (ICDE)*, pages 215–215. IEEE Computer Society.
- Pei, J., Wang, H., Liu, J., Wang, K., Wang, J., and Yu, P. S. (2006). Discovering frequent closed partial orders from strings. *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1467–1481.
- Peterson, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, Englewood Cliffs (NJ).
- Piernik, M., Brzezinski, D., Morzy, T., and Lesniewska, A. (2015). Xml clustering: a review of structural approaches. *The Knowledge Engineering Review*, 30:297–323.

- Pika, A., van der Aalst, W. M. P., Fidge, C. J., ter Hofstede, A. H. M., and Wynn, M. T. (2013). Profiling event logs to configure risk indicators for process delays. In Salinesi, C., Norrie, M., and Pastor, s., editors, *Advanced Information Systems Engineering*, volume 7908 of *Lecture Notes in Computer Science*, pages 465–481. Springer Berlin Heidelberg.
- Polpinij, J., Ghose, A., and Dam, H. K. D. (2010). Business rules discovery from process design repositories. In *Proceedings of the 6th World Congress on Services (SERVICES-1)*, pages 614–620, Miami, FL, USA.
- Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1(1):81–106.
- Quinlan, J. R. (1990). Learning logical definitions from relations. *Machine learning*, 5(3):239–266.
- Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*, volume 1. Morgan Kaufmann.
- Quinlan, J. R. and Cameron-Jones, R. M. (1993). FOIL: A midterm report. In Brazdil, P., editor, *Machine Learning: ECML-93*, volume 667 of *Lecture Notes in Computer Science*, pages 1–20. Springer Berlin Heidelberg.
- Raggett, D., Le Hors, A., Jacobs, I., et al. (1999). HTML 4.01 specification. *W3C recommendation*, 24.
- Rebuge, Á. and Ferreira, D. R. (2012). Business process analysis in healthcare environments: A methodology based on process mining. *Information Systems*, 37(2):99–116.
- Reijers, H. A. (2003). *Design and Control of Workflow Processes: Business Process Management for the Service Industry*. Springer-Verlag.
- Rogge-Solti, A. and Weske, M. (2013). Prediction of remaining service execution time using stochastic Petri nets with arbitrary firing delays. In Basu, S., Pautasso, C., Zhang, L., and Fu, X., editors, *Service-Oriented Computing*, volume 8274 of *Lecture Notes in Computer Science*, pages 389–403. Springer Berlin Heidelberg.
- Rozinat, A., de Jong, I. S. M., Gunther, C. W., and van der Aalst, W. M. P. (2009a). Process mining applied to the test process of wafer scanners in ASML. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 39(4):474–479.
- Rozinat, A., Mans, R. S., Song, M., and van der Aalst, W. M. P. (2009b). Discovering simulation models. *Information Systems*, 34(3):305–327.

- Rozinat, A. and van der Aalst, W. M. P. (2006). Decision mining in PROM. In *Proceedings of the 4th International Conference on Business Process Management, BPM '06*, pages 420–425, Berlin, Heidelberg. Springer-Verlag.
- Rückert, U. and Kramer, S. (2004). Frequent free tree discovery in graph data. In *Proceedings of the ACM symposium on Applied Computing*, pages 564–570. ACM.
- Sayal, M., Casati, F., Dayal, U., and Shan, M.-C. (2002). Business process cockpit. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*, pages 880–883. VLDB Endowment.
- Scheer, A.-W. (1994). *Business Process Engineering : Reference Models for Industrial Companies*. Springer, Berlin, 2 edition.
- Schimm, G. (2003). Mining most specific workflow models from event-based data. In van der Aalst, W. M. P. and Weske, M., editors, *Business Process Management*, volume 2678 of *Lecture Notes in Computer Science*, pages 25–40. Springer Berlin Heidelberg.
- Shahbazi, A. and Miller, J. (2014). Extended subtree: A new similarity function for tree structured data. *IEEE Transactions on Knowledge and Data Engineering*, 26(4):864–877.
- Shasha, D., Wang, J. T. L., and Zhang, S. (2004). Unordered tree mining with applications to phylogeny (ICDE'04). In *Proceedings of the 20th International Conference on Data Engineering*, pages 708–719.
- Sint, R., Schaffert, S., Stroka, S., and Ferstl, R. (2009). Combining unstructured, fully structured and semi-structured information in semantic wikis. In *Fourth Workshop on Semantic Wikis—The 6th European Semantic Web Conference (SemWiki '09)*.
- Sipos, R., Fradkin, D., Moerchen, F., and Wang, Z. (2014). Log-based predictive maintenance. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'14)*, pages 1867–1876, New York, NY, USA. ACM.
- Solé, M. and Carmona, J. (2010). Rbminer: A tool for discovering Petri nets from transition systems. In *Proceedings of the 8th International Conference on Automated Technology for Verification and Analysis (ATVA'10)*, pages 396–402, Berlin, Heidelberg. Springer-Verlag.
- Song, M. and van der Aalst, W. M. P. (2007). Supporting process mining by showing events at a glance. In *Proceedings of the 17th Annual Workshop on Information Technologies and Systems (WITS)*, pages 139–145.

- Sonnenberg, C. and vom Brocke, J. (2014). The missing link between BPM and Accounting. *Business Process Management Journal*, 20(2):213–246.
- Suciu, D. (2000). Semistructured data and XML. In Tanaka, K., Ghandeharizadeh, S., and Kambayashi, Y., editors, *Information Organization and Databases*, volume 579 of *The Springer International Series in Engineering and Computer Science*, pages 9–30. Springer US.
- Suriadi, S., Ouyang, C., van der Aalst, W. M. P., and ter Hofstede, A. H. M. (2013). Root cause analysis with enriched process logs. In La Rosa, M. and Soffer, P., editors, *Business Process Management Workshops*, volume 132 of *Lecture Notes in Business Information Processing*, pages 174–186. Springer Berlin Heidelberg.
- Swinnen, J., Depaire, B., Jans, M., and Vanhoof, K. (2012). A process deviation analysis - a case study. In Daniel, F., Barkaoui, K., and Dustdar, S., editors, *Business Process Management Workshops*, volume 99 of *Lecture Notes in Business Information Processing*, pages 87–98. Springer Berlin Heidelberg.
- Tagarelli, A. (2012). *XML Data Mining: Models, Methods, and Applications*. IGA GLOBAL.
- Tan, H., Dillon, T., Hadzic, F., Chang, E., and Feng, L. (2005). MB3-Miner: Efficiently mining embedded subtrees using tree model guided candidate generation. In *Proceedings of the First International Workshop on Mining Complex Data 2005 (MCD 2005)*, pages 103–110, Los Alamitos. IEEE Computer Society Press.
- Tan, H., Dillon, T. S., Hadzic, F., Chang, E., and Feng, L. (2006). IMB3-Miner: Mining induced/embedded subtrees by constraining the level of embedding. In Ng, W.-K., Kitsuregawa, M., Li, J., and Chang, K., editors, *Advances in Knowledge Discovery and Data Mining*, volume 3918 of *Lecture Notes in Computer Science*, pages 450–461. Springer Berlin Heidelberg.
- Tatikonda, S., Parthasarathy, S., and Kurc, T. (2006). TRIPS and TIDES: New algorithms for tree mining. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management (CIKM'06)*, pages 455–464, New York, NY, USA. ACM.
- ter Hofstede, A. H. M., van der Aalst, W. M. P., Adams, M., and Russell, N. (2010). *Modern Business Process Automation: YAWL and Its Support Environment*. Springer.
- Termier, A., Rousset, M.-C., and Sebag, M. (2002). TreeFinder: A first step towards XML data mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining, ICDM '02*, pages 450–457, Washington, DC, USA. IEEE.

- Termier, A., Rousset, M. C., Sebag, M., Ohara, K., Washio, T., and Motoda, H. (2008). DryadeParent, an efficient and robust closed attribute tree mining algorithm. *IEEE Transactions on Knowledge and Data Engineering*, 20(3):300–320.
- Thabtah, F. (2007). A review of associative classification mining. *The Knowledge Engineering Review*, 22(01):37–65.
- Thabtah, F. A., Cowling, P., and Peng, Y. (2004). MMAC: A new multi-class, multi-label associative classification approach. In *Proceedings of the Fourth IEEE International Conference on Data Mining (ICDM'04)*, pages 217–224. IEEE.
- Tiwari, A., Turner, C. J., and Majeed, B. (2008). A review of business process mining: State-of-the-art and future trends. *Business Process Management*, 14(1):5–22.
- Uno, T., Kiyomi, M., and Arimura, H. (2004). LCM ver.2: Efficient mining algorithms for Frequent/Closed/maximal itemsets. In *Proceeding of the First International Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*.
- Vaarandi, R. (2004). A breadth-first algorithm for mining frequent patterns from event logs. In Aagesen, F. A., Anutariya, C., and Wuwongse, V., editors, *Intelligence in Communication Systems*, volume 3283 of *Lecture Notes in Computer Science*, pages 293–308. Springer Berlin Heidelberg.
- Van der Aalst, W., Weijters, T., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.
- van der Aalst, W. M. P. (2009). Using process mining to generate accurate and interactive business process maps. In Abramowicz, W. and Flejter, D., editors, *Business Information Systems Workshops*, volume 37 of *Lecture Notes in Business Information Processing*, pages 1–14. Springer Berlin Heidelberg.
- van der Aalst, W. M. P. (2011a). *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer.
- van der Aalst, W. M. P. (2011b). Teleclaim Dataset. http://www.processmining.org/event_logs_and_models_used_in_book (This dataset is accompanied with the book (van der Aalst, 2011a). Last accessed 19 May 2015).
- van der Aalst, W. M. P. (2011c). Telephone Repair Dataset. http://www.processmining.org/event_logs_and_models_used_in_book (This dataset is accompanied with the book (van der Aalst, 2011a). Last accessed 19 May 2015).

- van der Aalst, W. M. P. (2013). Business process management: A comprehensive survey. *ISRN Software Engineering*, 2013:1–37.
- Van Der Aalst, W. M. P., Reijers, H. A., and Song, M. (2005). Discovering social networks from event logs. *Computer Supported Cooperative Work (CSCW)*, 14(6):549–593.
- van der Aalst, W. M. P., Rubin, V., Verbeek, H. M. W., van Dongen, B. F., Kindler, E., and Günther, C. W. (2010). Process mining: A two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87–111.
- van der Aalst, W. M. P., Schonenberg, M. H., and Song, M. (2011). Time prediction based on process mining. *Information Systems*, 36(2):450–475. Special Issue: Semantic Integration of Data, Multimedia, and Services.
- van der Aalst, W. M. P., ter Hofstede, A. H. M., Kiepuszewski, B., and Barros, A. P. (2003). Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51.
- van der Aalst, W. M. P. and van Dongen, B. F. (2002). Discovering workflow performance models from timed logs. In Han, Y., Tai, S., and Wikarski, D., editors, *Proceedings of the International Conference on Engineering and Deployment of Cooperative Information Systems*, volume 2480 of *Lecture Notes in Computer Science*, pages 45–63. Springer Berlin Heidelberg.
- van der Aalst, W. M. P., Weijters, T., and Maruster, L. (2004). Workflow mining: Discovering process models from event logs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1128–1142.
- van Dongen, B. (2012). BPI Challenge 2012.
- van Dongen, B. F. (2011). Real-life event logs - hospital logs. <http://dx.doi.org/10.4121/uuid:d9769f3d-0ab0-4fb8-803b-0d1120ffcf54>.
- van Dongen, B. F., de Medeiros, A. K. A., Verbeek, H. M. W., Weijters, A. J. M. M., and van der Aalst, W. M. P. (2005). The ProM framework: A new era in process mining tool support. In Ciardo, G. and Darondeau, P., editors, *Applications and Theory of Petri Nets 2005*, volume 3536 of *Lecture Notes in Computer Science*, pages 444–454. Springer Berlin Heidelberg.
- van Dongen, B. F. and van der Aalst, W. M. P. (2004). Multi-phase process mining: Building instance graphs. In Atzeni, P., Chu, W., Lu, H., Zhou, S., and Ling, T.-W., editors, *Conceptual Modeling ER 2004*, volume 3288 of *Lecture Notes in Computer Science*, pages 362–376. Springer Berlin Heidelberg.

- van Dongen, B. F. and van der Aalst, W. M. P. (2005). A meta model for process mining data. In Casto, J. and Teniente, E., editors, *Proceedings of the CAiSE'05 Workshops*, volume 2, pages 309–320. FEUP, Porto, Portugal.
- Vasilyev, E., Ferreira, D. R., and Iijima, J. (2013). Using inductive reasoning to find the cause of process delays. In *Proceedings of the 15th Conference on Business Informatics (CBI)*, pages 242–249. IEEE.
- Veloso, A., Meira Jr., W., and Zaki, M. J. (2006). Lazy associative classification. In *Proceedings of the Sixth International Conference on Data Mining (ICDM'06)*, pages 645–654, Washington, DC, USA. IEEE Computer Society.
- Verbeek, H. M. W., Buijs, J. C. A. M., van Dongen, B. F., and van der Aalst, W. M. P. (2011). XES, XESame, and ProM 6. In Soffer, P. and Proper, E., editors, *Information Systems Evolution*, volume 72 of *Lecture Notes in Business Information Processing*, pages 60–75. Springer Berlin Heidelberg.
- Vishwanathan, S. V. N. and Smola, A. J. (2004). Fast kernels for string and tree matching. *Kernel Methods in Computational Biology*, pages 113–130.
- Wang, C., Hong, M., Pei, J., Zhou, H., Wang, W., and Shi, B. (2004). Efficient pattern-growth methods for frequent tree pattern mining. In Dai, H., Srikant, R., and Zhang, C., editors, *Advances in Knowledge Discovery and Data Mining*, volume 3056 of *Lecture Notes in Computer Science*, pages 441–451. Springer Berlin Heidelberg.
- Wang, K. and Liu, H. (1998). Discovering typical structures of documents: A road map approach. In *Proceedings of the 21st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'98)*, pages 146–154, New York, NY, USA. ACM.
- Wang, K., Zhou, S., and He, Y. (2000). Growing decision trees on support-less association rules. In *Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'00)*, pages 265–269, New York, NY, USA. ACM.
- Wang, S., Hong, Y., and Yang, J. (2012). XML document classification using closed frequent subtree. In Bao, Z., Gao, Y., Gu, Y., Guo, L., Li, Y., Lu, J., Ren, Z., Wang, C., and Zhang, X., editors, *Web-Age Information Management*, volume 7419 of *Lecture Notes in Computer Science*, pages 350–359. Springer Berlin Heidelberg.
- Weijters, A. J. M. M. and van der Aalst, W. M. P. (2003). Rediscovering workflow models from event-based data using little thumb. *Integrated Computer-Aided Engineering*, 10(2):151–162.

- Weske, M. (2007). *Business Process Management - Concepts, Languages, Architectures*, volume 14. Springer.
- Wu, X., Kumar, V., Ross Quinlan, J., Ghosh, J., Yang, Q., Motoda, H., McLachlan, G. J., Ng, A., Liu, B., Yu, P. S., Zhou, Z.-H., Steinbach, M., Hand, D. J., and Steinberg, D. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, 14(1):1–37.
- Xiao, Y., Yao, J.-F., Li, Z., and Dunham, M. H. (2003). Efficient data mining for maximal frequent subtrees. In *Proceedings of the Third IEEE International Conference on Data Mining (ICDM'03)*, pages 379–386, Washington, DC, USA. IEEE Computer Society.
- Xu, W., Huang, L., Fox, A., Patterson, D., and Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles (SOSP'09)*, pages 117–132, New York, NY, USA. ACM.
- Yan, X. and Han, J. (2002). gSpan: Graph-based substructure pattern mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, pages 721–724, Washington, DC, USA. IEEE Computer Society.
- Yan, X., Han, J., and Afshar, R. (2003). CloSpan: Mining closed sequential patterns in large datasets. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SDM'03)*, pages 166–177, San Francisco, CA, USA. SIAM.
- Yang, W.-S. and Hwang, S.-Y. (2006). A process-mining framework for the detection of healthcare fraud and abuse. *Expert Systems with Applications*, 31(1):56–68.
- Yin, X. and Han, J. (2003). CPAR: Classification based on predictive association rules. In *Proceedings of the 2003 SIAM International Conference on Data Mining (SDM'03)*, volume 3, pages 369–376, San Francisco, CA, USA. SIAM.
- Zaki, M. J. (2000). Scalable algorithms for association mining. *IEEE Transactions on Knowledge and Data Engineering*, 12(3):372–390.
- Zaki, M. J. (2002). Efficiently mining frequent trees in a forest. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 71–80, Edmonton, Alberta, Canada. ACM.
- Zaki, M. J. (2005a). Efficiently mining frequent embedded unordered trees. *Fundamenta Informaticae*, 66(1):33–52.

- Zaki, M. J. (2005b). Efficiently mining frequent trees in a forest: Algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17:1021–1035.
- Zaki, M. J. and Aggarwal, C. C. (2006). XRules: An effective algorithm for structural classification of XML data. *Machine Learning*, 62(1-2):137–170.
- Zhang, S. and Wang, J. T.-L. (2008). Discovering frequent agreement subtrees from phylogenetic data. *IEEE Transactions on Knowledge and Data Engineering*, 20(1):68–82.

Every reasonable effort has been made to acknowledge the owners of copyright material. I would be pleased to hear from any copyright owner who has been omitted or incorrectly acknowledged.