Frequent Subgraph Mining Algorithms on Weighted Graphs

Thesis submitted in accordance with
the requirements of the University of Liverpool for
the degree of Doctor in Philosophy

by

Chuntao Jiang

April 2011

Abstract

This thesis describes research work undertaken in the field of graph-based knowledge discovery (or graph mining). The objective of the research is to investigate the benefits that the concept of *weighted* frequent subgraph mining can offer in the context of the graph model based classification. Weighted subgraphs are graphs where some of the vertexes/edges are considered to be more significant than others. How to discover frequent sub-structures with different strengths is the main issue to be resolved in this thesis. The main approach to addressing this issue is to integrate weight constraints into the frequent subgraph mining process. It is suggested that the utilization of weighted frequent subgraph mining generates more discriminate and significant subgraphs, which will have application in, for example, the classification and clustering of graph data.

To my parents and my sister

“穷则独善其身，达则兼善天下。”

《孟子. 尽心上》

“积土成山，风雨兴焉；积水成渊，蛟龙生焉；积善成德，而神明自得，圣心备焉。故不积跬步，无以至千里；不积小流，无以成江海。骐骥一跃，不能十步；驽马十驾，功在不舍。锲而舍之，朽木不折；锲而不舍，金石可镂。蚯无爪牙之利，筋骨之强，上食埃土，下饮黄泉，用心一也。蟹六跪而二螯，非蛇鳝之穴无可寄托者，用心躁也。”

《荀子. 劝学》

Acknowledgement

First and Foremost, I am very grateful to my first supervisor Dr. Frans Coenen, specially for his constant patience, support and encouragement throughout my four-year study in the Department.

I am specially thankful to Dr. Michele Zito, who agreed to act as my second supervisor and reviewed many pieces of my writings. I also thank him for providing many constructive suggestions about my research work.

I would also like to thank my colleague, Ashraf EI Sayed, for our collaborative research work. I am also obliged to my friends who have given me a good time during our regular meetings. They are: Yanbo Wang, Jamie McAuslane and Judy McAuslane.

Finally, I want to send my special regards to my father (Xinshi Jiang). Without his emotional and financial support, I would never have had the chance to take a first step in pursuit of my academic career.

March 2011
Liverpool, England

Contents

Abstract i

Acknowledgement iii

List of Figures xi

List of Tables xv

1 Introduction 1

1.1 Research Motivation . . . 3
1.2 Research Question . . . 7
1.3 Methodology . . . 8
1.4 Evaluation Criteria . . . 9
1.5 Contribution . . . 10
1.6 Organization of the Thesis . . . 11

2 Background 13

2.1 Concepts . . . 13
2.2 Problem Definition . . . 17
2.3 Graph Isomorphism Detection . . . 18
2.4 Overview of Frequent Subgraph Mining . . . 21
2.4.1 Canonical representations . . . 23
2.4.2 Frequency measures . . . 27
2.4.3 Candidate generation . . . 28
2.4.3.1 Rightmost path expansion . . . 28
2.4.3.2 Equivalence class based extension . . . 29
2.4.3.3 Right-and-left tree join . . . 30
2.4.3.4 Extension and join . . . 30
2.4.3.5 Level-wise join . . . 31
2.5 Frequent Subtree Mining Algorithms . . . 32
2.5.1 Rooted unordered tree mining . . . 32
2.5.2 Rooted ordered tree mining . . . 34

2.5.3 Free tree mining . . . 37
2.5.4 Hybrid tree mining . . . 38
2.5.4.1 Rooted ordered or unordered trees mining . . . 38
2.5.4.2 Rooted unordered or free trees mining . . . 38
2.5.5 Summary of frequent subtree mining algorithms . . . 39
2.6 Frequent Subgraph Mining Algorithms . . . 43
2.6.1 General purpose frequent subgraph mining . . . 44
2.6.1.1 Inexact match based . . . 44
2.6.1.2 Exact match based . . . 45
2.6.1.2.1 Transaction graph based mining . . . 46
2.6.1.2.2 Single graph based mining . . . 50
2.6.2 Pattern dependent frequent subgraph mining . . . 52
2.6.2.1 Mining relational patterns . . . 52
2.6.2.2 Mining maximal patterns and closed patterns . . . 53
2.6.2.2.1 Mining maximal patterns . . . 53
2.6.2.2.2 Mining closed patterns . . . 54
2.6.2.3 Mining clique patterns . . . 55
2.6.2.4 Mining constrained patterns . . . 56
2.6.3 Summary . . . 56
2.7 Classification Using Frequent Subgraphs . . . 60
2.8 Social Network Analysis . . . 62
2.9 Weighted Frequent Subgraph Mining . . . 63
2.10 Summary . . . 66
2.10.1 Pseudo-codes of gSpan . . . 67
2.10.2 Pseudo-codes of FFSM . . . 67
2.10.3 Pseudo-codes of GASTON . . . 67

3 Graph Data Sets **71**
3.1 Synthetic Data Sets . . . 71
3.1.1 ST1 - Synthetic trees created by a random tree generator . . . 72
3.1.2 ST2 - Synthetic images represented by quad-trees . . . 72
3.2 Real Datasets . . . 74
3.2.1 RT1 - Web usage mining scenario . . . 74
3.2.2 RT2 - MRI brain scan images represented by quad-trees . . . 75
3.2.3 RT3 - Text mining founded on document semantics . . . 77
3.2.4 RG1 - Chemical compound analysis scenario . . . 79
3.2.5 RG2 - Mammographic images represented by ARGs . . . 80
3.2.6 RG3 - Photographic images represented by ARGs . . . 82
3.2.7 RG4 - Text mining founded on term occurrence . . . 82
3.2.8 RG5 - Social network mining scenario . . . 84

3.3 Summary of Data 86

4 Weighting Functions for Vertexes and Edges in Graphs 89
4.1 Structural Weighting Function 90
4.1.1 SW1 - Normalized occurrences based method 90
4.1.2 SW2 - Phi correlation coefficient based method 91
4.1.3 SW3 - Normalized mutual information based method 92
4.1.4 SW4 - Mutual information based method 94
4.1.5 SW5 - Point-wise mutual information based method 95
4.2 Content Weighting Function 96
4.2.1 CW1 - User predefined weights 96
4.2.2 Calculation of edge weights using class labels 96
4.2.2.1 CW2 - Calculation of edge weights using χ^2 values . . . 97
4.2.2.2 CW3 - Calculation of edge weights using NMI values . 99
4.3 Summary 100

5 Subgraph Weighting Schemes That Maintain the DCP 103
5.1 Overview 103
5.2 Weighting Schemes Using Vertex or Edge Weights 104
5.2.1 Average Total Weighting (ATW) scheme 105
5.2.1.1 Pseudo-codes of ATW 106
5.2.2 Affinity Weighting (AW) scheme 107
5.2.2.1 Pseudo-codes of AW 108
5.2.3 Correlation Measures based Weighting (CMW) scheme 109
5.2.3.1 Pseudo-codes of CMW 110
5.3 Weighting Schemes That Do Not Use Vertex or Edge Weights 111
5.3.1 Jaccard Similarity based Weighting (JSW) scheme 111
5.3.1.1 Pseudo-codes of JSW 112
5.4 Summary 113

6 Experimental Study 115
6.1 Implementation 116
6.2 Overview of the Data 116
6.3 The Evaluation of the ATW scheme 118
6.3.1 The evaluation of the ATW scheme on trees 118
6.3.2 The evaluation of the ATW scheme on undirected graphs 121
6.3.3 The evaluation of the ATW scheme on directed graphs 125
6.3.4 Summary & discussion 127
6.4 The Evaluation of the AW scheme 128
6.4.1 The evaluation of the AW scheme on trees 128

6.4.2 The evaluation of the AW scheme on undirected graphs 131
6.4.3 The evaluation of the AW scheme on directed graphs 132
6.4.4 Summary & discussion 133
6.5 The Evaluation of the CMW scheme 134
6.5.1 The evaluation of the CMW scheme on trees 135
6.5.2 The evaluation of the CMW scheme on undirected graphs 138
6.5.3 The evaluation of the CMW scheme on directed graphs 141
6.5.4 Summary & discussion 143
6.6 The Evaluation of the JSW scheme 144
6.6.1 The evaluation of the JSW scheme on trees 144
6.6.2 The evaluation of the JSW scheme on undirected graphs 147
6.6.3 The evaluation of the JSW scheme on directed graphs 149
6.6.4 Summary & discussion 151
6.7 Summary 152

7 Two Case Studies **153**
7.1 Case Study 1 - The RT2 Data 153
7.1.1 Efficiency test 154
7.1.2 Effectiveness of the patterns 156
7.1.3 Summary 159
7.2 Case Study 2 - The RT3 Data 159
7.2.1 Efficiency test 159
7.2.2 Effectiveness of the patterns 162
7.2.3 Summary 163

8 Subgraph Weighting Schemes That Do Not Maintain the DCP **165**
8.1 Utility Based Weighting (UBW) Scheme 165
8.1.1 Pseudo-codes of UBW 168
8.2 Experimental Study 169
8.2.1 The evaluation of the UBW scheme on trees 170
8.2.2 The evaluation of the UBW scheme on undirected graphs 173
8.2.3 The evaluation of the UBW scheme on directed graphs 176
8.3 Summary and Discussion 178

9 Further Discussions **181**
9.1 The Usage of Weighting Functions 181
9.2 The Usage of Subgraph Weighting Schemes 182

10 Conclusion **191**
10.1 Findings 191
10.2 Contributions 193

10.3 Future Directions 194

Bibliography **219**

A Graph File Formats **221**

A.1 GraphML Format 221

A.2 Simple LineGraph Format 222

B Additional Experimental Results **225**

B.1 The Experimental Results for Trees 225

B.1.1 The application of the ATW scheme 225

B.1.1.1 The ST1, ST2, and RT1 data 225

B.1.1.2 The RT2 data 225

B.1.2 The application of the AW scheme 227

B.1.2.1 The ST1, ST2, and RT1 data 227

B.1.2.2 The RT2 data 231

B.1.3 The application of the CMW scheme 232

B.1.3.1 The ST1 and RT1:CSLOGS-ALL data 233

B.1.3.2 The ST2 and RT1:CSLOGS-1(2) data 235

B.1.4 The application of the JSW scheme 235

B.1.4.1 The ST1 data 236

B.1.4.2 The ST2 data 236

B.1.4.3 The RT1 data 236

B.1.4.4 The RT2 data 237

B.1.5 The application of the UBW scheme 238

B.2 The Experimental Results for Undirected Graphs 242

B.2.1 The application of the ATW scheme 242

B.2.1.1 The RG1 data 242

B.2.1.2 The RG2 and RG3 data 243

B.2.2 The application of the AW scheme 243

B.2.2.1 The RG1 data 243

B.2.3 The application of the CMW scheme 246

B.2.3.1 The RG1:CH2 data 246

B.2.3.2 The RG1:CH1, RG2, and RG3 data 247

B.2.4 The application of the JSW scheme 248

B.2.4.1 The RG1 data 248

B.2.5 The application of the UBW scheme 249

B.3 The Experimental Results for Directed Graphs 249

B.3.1 The application of the ATW scheme 250

B.3.1.1 The RG4 data 250

B.3.2 The application of the CMW scheme 250
B.3.2.1 The RG4 data 250
B.3.2.2 The RG5 data 251
B.3.3 The application of the JSW scheme 254
B.3.3.1 The RG5 data 254

C Published Work **257**

List of Figures

1.1 Examples of graph represented data. 2
1.2 Evaluation model for weighted frequent subgraph mining 10

2.1 Different types of trees 16
2.2 Lattice($\mathbb{GD}$) 17
2.3 Two types of FSM approaches. Note that the subgraph lattice is shown "upside-down". Vertexes corresponding to graphs with fewer edges are displayed at the top of the picture in each case. 21
2.4 A graph example with its adjacency matrix 24
2.5 An example of two types of free trees 26
2.6 A tree example 27
2.7 An illustration of rightmost path expansion 29
2.8 An illustration of right-and-left tree join 30
2.9 Frequent pattern based classification 62

3.1 An illustration of the quad-tree representation. The image on the left is recursively divided into sub-quadrants (NW, NE, SW and SE). 73
3.2 An illustration of quad-tree represented random images 73
3.3 The procedure for generating quad-tree represented MRI brain scan images 76
3.4 An example of semantic graph based representation 78
3.5 The procedure for building image interest points based graphs. 80
3.6 An example of interest points identified by a Harris-Affine detector. The yellow points on (b) denote the rich information obtained by the detector on (a). 81
3.7 A second example of images and their interest points. The raw images belong to two different classes: 'Bonsai' and 'Sunflower'. Interests points, identified by a Harris-Affine detector, are shown in (b) and (d) with yellow points. 83
3.8 Procedure for representing documents as graphs. 84
3.9 The process of building graphs from the CTS database 86

4.1 Normalized occurrences based method example 91

4.2 PCC based method example 92
4.3 NMI based method example 93
4.4 Mutual information based method example 95
4.5 PMI based method example 96
4.6 χ^2 based method example 98
4.7 An example of computing edge weights using NMI values 100

5.1 An example of calculating weights by the ATW scheme 106
5.2 An example of calculating weights by the AW scheme 108
5.3 An example of computing weights by the JSW scheme 112

7.1 The performance of gSpan-JSW with different γ values on the RT2 data 157
7.2 The performance of gSpan-ATW on the RT3 data 160
7.3 The performance of gSpan-AW on the RT3 data 160
7.4 The performance of gSpan-CMW on the RT3 data 161
7.5 The performance of gSpan-JSW on the RT3 data 162

8.1 An example of computing the overlap similarity 166
8.2 An example of computing share values. 167

A.1 A graph example 221
A.2 A LineGraph format illustration 223

B.1 The performance of gSpan-ATW on the ST2 data 226
B.2 The performance of gSpan-ATW on the QT-D5 data 228
B.3 The performance of gSpan-ATW on the QT-D6 data 228
B.4 The performance of gSpan-ATW on the QT-D7 data 229
B.5 The performance of gSpan-AW on the IM1000-D4 data 230
B.6 The performance of gSpan-AW on the IM1000-D5 and IM1000-D6 data 230
B.7 The performance of gSpan-AW on the CSLOGS-1 data 231
B.8 The performance of gSpan-AW on the CSLOGS-2 data 231
B.9 The performance of gSpan-AW on the RT2 data 233
B.11 The performance of gSpan-JSW on the CSLOGS-ALL data 236
B.10 The performance of gSpan-JSW on the ST2 data 237
B.12 The performance of gSpan-UBW on the CSLOGS-ALL data 239
B.13 The performance of gSpan-UBW on the CSLOGS-1 data 239
B.14 The performance of gSpan-UBW on the CSLOGS-2 data 240
B.15 The performance of gSpan-UBW on the QT-D5 data 240
B.16 The performance of gSpan-UBW on the QT-D6 data 240
B.17 The performance of gSpan-UBW on the QT-D7 data 241
B.18 The performance of gSpan-ATW on the CH1 data 242

B.19 The performance of gSpan-ATW on the CH2 data 243
B.20 The performance of gSpan-AW on the CH1 data 244
B.21 The performance of gSpan-AW on the CH2 data 246
B.22 The performance of gSpan-CMW on the CH1 data 247
B.23 The performance of gSpan-JSW on the CH1 data 248
B.24 The performance of gSpan-JSW on the CH2 data 249
B.25 The performance of gSpan-CMW using CW2 and CW3 on the IMDB data250
B.26 The performance of gSpan-CMW using CW2 and CW3 on the Amazon & Ohsumed data 251
B.27 The performance of gSpan-CMW on the Lancashire data 253
B.28 The performance of gSpan-CMW on the Scotland data 253
B.29 The performance of gSpan-CMW on the GB data 253
B.30 The performance of gSpan-JSW on the Lancashire data 254
B.31 The performance of gSpan-JSW on the Scotland data 254
B.32 The performance of gSpan-JSW on the GB data 255

List of Tables

1.1 Summary of graph data sets 9

2.1 Basic vocabulary 14
2.2 Notation used throughout the thesis 18
2.3 Overview of a number of exact matching (sub)graph isomorphism algorithms 20
2.4 Taxonomy of frequent subtree mining algorithms 31
2.5 Main techniques used by frequent subtree mining algorithms 39
2.6 A summary of frequent subtree mining algorithms 40
2.7 General purpose frequent subgraph mining algorithm categorisation . . 44
2.8 Categorisation of pattern dependent frequent subgraph mining algorithms 52
2.9 A summary of the FGM algorithms reported in this thesis 57
2.10 Main techniques used by FGM algorithms reported in this thesis 58

3.1 Synthetic tree data set parameters 72
3.2 Characteristics of the synthetic "web usage" data sets 72
3.3 Characteristics of quad-tree represented image data sets 73
3.4 Properties of tree based CSLOGS web usage data sets 75
3.5 Properties of RT2 data at different quad-tree levels 76
3.6 Categorisation of the text data sets according to the graph representation method adopted 77
3.7 Properties of tree represented Medline documents 79
3.8 Properties of chemical compound graph data sets 80
3.9 Properties of graph represented mammographic images 82
3.10 Properties of RG3 data using different vocabularies 82
3.11 Properties of graph data sets defined using the term occurrences based representation 85
3.12 Properties of graphs generated using the CTS database 86
3.13 Summary of the graphic data employed in this thesis 87

4.1 A two-way contingency table of e_i and c_j 97
4.2 Two-way contingency tables for 'a' and class labels 98

4.3 Two-way contingency tables for 'e' and class labels 99

6.1 A summary of data sets employed throughout this chapter 117
6.2 The performance of gSpan-ATW using SW1 on the ST1, ST2, and RT1 data 119
6.3 The accuracy of the classifiers using patterns discovered by gSpan-ATW using SW1 on the ST2 and RT1:CSLOGS-1(2) data 121
6.4 The performance of gSpan-ATW using SW1 on the RG1 data 122
6.5 The performance of gSpan-ATW using CW1-E on the RG2 and RG3 data 122
6.6 The accuracy of the classifiers using patterns discovered by gSpan-ATW using SW1 on the CH1 data 123
6.7 The accuracy of the classifiers using patterns discovered by gSpan-ATW using SW4 and SW5 on the CH1 data 124
6.8 The accuracy of the classifiers using patterns discovered by gSpan-ATW using CW1-E on the RG2 and RG3 data 124
6.9 The performance of gSpan-ATW using CW1 on the RG4 data 125
6.10 The performance of gSpan-ATW using CW1-E and SW5 on the RG5 data 126
6.11 The accuracy of the classifiers using patterns discovered by gSpan-ATW using CW1-E on the RG4 data 127
6.12 The performance of gSpan-AW using SW1 on the ST1, ST2, and RT1:CSLOGS-1(2) data 129
6.13 The accuracy of the classifiers using patterns discovered by gSpan-AW with SW1 on the ST2 and RT1:CSLOGS-1(2) data 130
6.14 The performance of gSpan-AW using SW1 on the CH2 data 131
6.15 The performance of of gSpan-AW with CW1-E on the RG3:BS-V500 data 132
6.16 The accuracy of the classifiers using patterns discovered by gSpan-AW using CW1-E on the RG3:BS-V500 data 132
6.17 The performance of gSpan-AW using CW1-E on the RG5 data 133
6.18 The performance of gSpan-CMW using SW2 on the ST1 and RT1:CSLOGS-ALL data 135
6.19 The performance of gSpan-CMW using CW2 on the ST2 and RT1:CSLOGS-1(2) data 136
6.20 The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the ST2 and RT1:CSLOGS-1(2) data 137
6.21 The performance of gSpan-CMW with CW2 on the RG1:CH1, RG2, and RG3 data 139
6.22 The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the CH1 data 140
6.23 The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the RG2 and RG3 data 140

6.24 The performance of gSpan-CMW using SW2 and SW3 on the RG4 and RG5 data . . . 141
6.25 The accuracy of the classifiers using patterns discovered by gSpan-CMW with SW2 on the RG4 data . . . 142
6.26 The performance of gSpan-JSW on the ST1 and RT1:CSLOGS-ALL data 145
6.27 The performance of gSpan-JSW on the ST2 and RT1:CSLOGS-1(2) data 146
6.28 The accuracy of the classifiers using patterns discovered by gSpan-JSW on the ST2 and RT1:CSLOGS-1(2) data . . . 146
6.29 The performance of gSpan-JSW on the RG1, RG2, and RG3 data . . . 148
6.30 The accuracy of the classifiers using patterns discovered by gSpan-JSW on the CH1 data . . . 148
6.31 The accuracy of the classifiers using patterns discovered by gSpan-JSW on the RG2 and RG3 data . . . 149
6.32 The performance of gSpan-JSW on the RG4 and RG5 data . . . 150
6.33 The accuracy of the classifiers using patterns discovered by gSpan-JSW on the RG4 data . . . 151

7.1 A summary of the RT2 data . . . 154
7.2 The performance of gSpan-ATW using SW1 on the RT2 data . . . 154
7.3 The performance of gSpan-ATW using SW4 on the RT2 data . . . 155
7.4 The performance of gSpan-JSW with $\gamma = 0.2$ on the RT2 data . . . 156
7.5 The accuracy of the classifiers using patterns discovered by gSpan-ATW with SW1 on the RT2 data . . . 156
7.6 The accuracy of the classifiers using patterns discovered by gSpan-ATW with SW4 on the RT2 data . . . 158
7.7 The accuracy of the classifiers using patterns discovered by gSpan-JSW with different γ values on the RT2 data . . . 158
7.8 A summary of the RT3 data . . . 159
7.9 The accuracy of the classifiers using patterns discovered by the weighted FSM algorithms on the RT3 data . . . 162

8.1 The performance of gSpan-UBW with SW1 on the tree data . . . 171
8.2 The performance of gSpan-UBW with SW4 on the tree data . . . 172
8.3 The accuracy of the classifiers using patterns discovered by gSpan-UBW using SW1 on the tree data . . . 173
8.4 The performance of gSpan-UBW with SW1 on the RG1 data . . . 174
8.5 The performance of gSpan-UBW using SW4 on the RG1 data . . . 174
8.6 The performance of gSpan-UBW with CW1-E on the RG2 and RG3 data 175
8.7 The accuracy of the classifiers using patterns discovered by gSpan-UBW with SW1 on the CH1 data . . . 175

8.8 The accuracy of the classifiers using patterns discovered by gSpan-UBW with CW1-E on the RG2 and RG3 data 176
8.9 The performance of gSpan-UBW with CW1-E on the RG4 and RG5 data 177
8.10 The accuracy of the classifiers using patterns discovered by gSpan-UBW with CW1-E on the RG4 data 178

9.1 The applicability of subgraph weighting schemes with respect to various data sets 182
9.2 The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the ST2 data 184
9.3 The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the CSLOGS-1(2) data 184
9.4 The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RT2 data 185
9.5 The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RT3 data 186
9.6 The AUC scores of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG1:CH1 data 186
9.7 The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG2 data 187
9.8 The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG3 data 187
9.9 The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG4 data 188

B.1 The performance of gSpan-ATW using SW4 and SW5 on the ST1, ST2, and RT1 data 227
B.2 The accuracy of the classifiers using patterns discovered by gSpan-ATW with SW4 and SW5 on the ST2 and RT1:CSLOGS-1(2) data 228
B.3 The performance of gSpan-AW using SW4 on the ST1, ST2, and RT1:CSLOGS-1(2) data 229

B.4 The accuracy of the classifiers using patterns discovered by gSpan-AW with SW4 on the ST2 and RT1:CSLOGS-1(2) data 232
B.5 The accuracy of the classifiers using patterns discovered by the standard FSM algorithms on the RT2 data 232
B.6 The accuracy of the classifiers using patterns discovered by gSpan-AW with SW1 and SW4 on the RT2 data 234
B.7 The performance of gSpan-CMW using SW3 on the ST1 and RT1:CSLOGS-ALL data 234
B.8 The performance of gSpan-CMW using CW3 on the ST2 data 234
B.9 The performance of gSpan-CMW using CW3 on the RT1:CSLOGS-1(2) data 235
B.10 The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW3 on the ST2 and RT1:CSLOGS-1(2) data 235
B.11 The performance of gSpan-JSW with different γ values on the ST1 data 236
B.12 The performance of gSpan-JSW with $\gamma = 5$ on the CSLOGS-1(2) data . 238
B.13 The performance of gSpan-JSW with $\gamma = 0.25$ on the RT2 data 238
B.14 The accuracy of the classifiers using patterns discovered by gSpan-UBW with SW4 on the tree data 241
B.15 The performance of gSpan-ATW using SW4 and SW5 on the RG1 data 242
B.16 The AUC scores of the classifiers using patterns discovered by gSpan-ATW with SW1 on the CH1 data 243
B.17 The AUC scores of the classifiers using patterns discovered by gSpan-ATW with SW4 and SW5 on the CH1 data 243
B.18 The performance of gSpan-ATW using CW1-N on the RG2 and RG3 data 244
B.19 The accuracy of the classifiers using patterns discovered by gSpan-ATW with CW1-N on the RG2 and RG3 data 244
B.20 The performance of the classifiers using patterns discovered by the standard FSM algorithms on the CH1 data 245
B.21 The performance of the classifiers using patterns discovered by gSpan-AW with SW1 on the CH1 data 245
B.22 The performance of the classifiers using patterns discovered by gSpan-AW with SW4 on the CH1 data 245
B.23 The performance of gSpan-AW using SW4 on the CH2 data 245
B.24 The performance of gSpan-CMW using SW2 on the CH2 data 246
B.25 The performance of gSpan-CMW using SW3 on the CH2 data 246
B.26 The performance of gSpan-CMW using CW3 on the RG1:CH1, RG2 and RG3 data 247
B.27 The AUC scores of the classifiers using patterns discovered by gSpan-CMW with CW2 on the CH1 data 247

B.28 The performance of the classifiers using patterns discovered by gSpan-CMW with CW3 on the CH1 data 248
B.29 The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW3 on the RG2 and RG3 data 248
B.30 The AUC scores of the classifiers using patterns discovered by gSpan-JSW on the CH1 data 249
B.31 The AUC scores of the classifiers using patterns discovered by gSpan-UBW with SW1 on the CH1 data 249
B.32 The performance of the classifiers using patterns discovered by gSpan-UBW with SW4 on the CH1 data 250
B.33 The accuracy of the classifiers using patterns discovered by gSpan-ATW with CW1-N on the RG4 data 250
B.34 The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the RG4 data 252
B.35 The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW3 on the RG4 data 252
B.36 The accuracy of the classifiers using patterns discovered by gSpan-CMW with SW3 on the RG4 data 252

Chapter 1

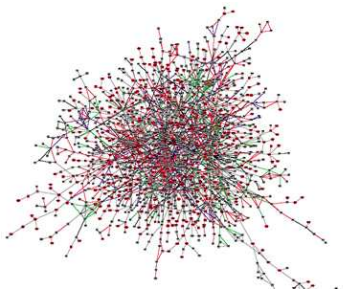
Introduction

The primary goal of data mining is to extract hidden, but useful, knowledge from data [Han and Kamber, 2006, Chen et al., 1996]. The data to which data mining may be applied can be categorised according to its representation mechanism: vectors, tables, texts, images, and so on. Data can also be categorized as being structured (e.g. molecule data), semi-structured (e.g. XML document collections) and non-structured (e.g. sound or video). Structured data is intuitively suited to graph representations. Common examples of structured data represented as graphs (Fig. 1.1) included protein-protein interaction networks, chemical compound structure graphs, bibliography graphs and behaviour graphs. Protein-protein interaction networks (Fig. 1.1(a)), representing the biological function occurring between at least two binding proteins, are commonly constructed by a set of vertexes representing proteins, connected by a set of edges representing direct physical interactions or function associations [Alm and Arkin, 2003]. Fig. 1.1(b) shows an example of a chemical compound graph (a molecule graph). The figure actually shows the molecular structure of the chemical compound 'Flucytosine' represented as an undirected graph where each vertex denotes an atom type and each edge denotes a bond type. In Fig. 1.1(c) a partial graph to model bibliography data is presented where 'a2' and 'a3' denote authors, 'w1', 'w2' and 'w3' denote publications, 'j2' denotes a journal, and 't1' indicates time of publication. A partial behaviour graph is a tool used to model software program execution (Fig. 1.1(d)), where each vertex indicates a functional module and each edge the relation between modules. There are many more examples.

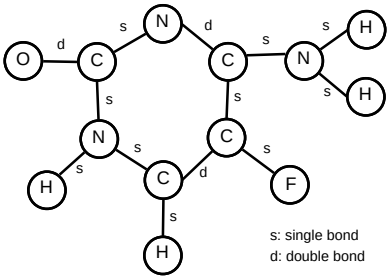
Because of the ease with which structural data can be represented using a graph format, substantial research effort has been directed towards the mining of graph data (also referred to as graph based data mining or graph mining). Examples include:

(1) Frequent subgraph mining [Dehaspe et al., 1998, Cook and Holder, 1994, 2000, Inokuchi et al., 2000, Kuramochi and Karypis, 2001, Huan et al., 2003, Borgelt and Berthold, 2002, Yan and Han, 2002, Nijssen and Kok, 2004].

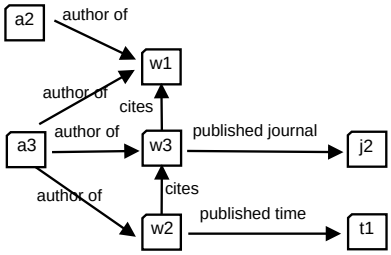
(2) Optimal graph pattern mining [Fan et al., 2008, Yan et al., 2008].



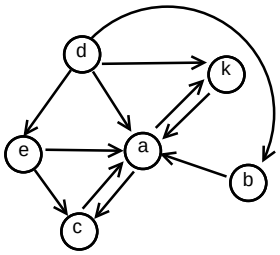
(a) Interaction network[Uetz, 2003]



(b) Molecule graph



(c) Bibliography graph



(d) Behaviour graph

Figure 1.1: Examples of graph represented data.

(3) Correlated graph pattern mining [Ke et al., 2007, Ozaki and Ohkawa, 2008, Ke et al., 2009].

(4) Graph pattern summarization [Xin et al., 2006b, Hasan et al., 2007, Chen et al., 2008].

(5) Approximate graph pattern mining [Kelley et al., 2003, Sharan et al., 2005, Chen et al., 2007b].

(6) Graph classification [Huan et al., 2004a, Kudo et al., 2004, Deshpande et al., 2005].

(7) Graph clustering [Flake et al., 2004, Newman, 2004b, Huang and Lai, 2006, Tsuda and Kudo, 2006].

(8) Graph indexing [Shasha et al., 2002, Yan et al., 2004, 2005a].

(9) Graph searching [Yan et al., 2005b, 2006a,b, Chen et al., 2007a].

The essence of graph mining is to extract useful knowledge from graph represented data by using techniques from fields such as data mining, machine learning, statistics, pattern recognition and graph theory. The importance of graph mining is reflected in the wide variety of domains to which graph mining (in all its forms) has been applied. Reported examples include: (a) chemical compound analysis [Deshpande et al., 2005, Fatta and Berthold, 2005, Wale and Karypis, 2006], (b) biological network analysis [Hu et al., 2005], (c) computer vision [Nowozin et al., 2007, Saigo et al., 2008], (d) work-flow mining [Greco et al., 2005], (e) social network mining [Freeman, 1979, Cai et al., 2005],

(f) link mining [Kleinberg, 1998, Brin and Page, 1998, Chakrabarti et al., 1999, Kosala and Blockeel, 2000, Getoor and Diehl, 2005, Liu, 2008], and (g) graph kernels [Gärtner et al., 2003, Kashima et al., 2003, Borgwardt and Kriegel, 2005, Ralaivola et al., 2005].

Frequent subgraph mining is one of the most common topics of graph mining. Generally, frequent subgraph mining aims to identify all subgraph patterns whose occurrences within a graph data set are above some user defined threshold. These subgraph patterns are called frequent subgraphs. The number of occurrences (the frequency) of each subgraph pattern is computed by a *support* measure. Theoretically, frequent subgraph mining can be formulated as a search in a search space, modelled by a lattice, consisting of all possible subgraph patterns. Because the number of possible frequent subgraphs increases exponentially with the size of the graph, completely traversing the search space is computational intractable, because of a "combinatorial explosion". Most frequent subgraph mining algorithms thus adopt a user specified *support threshold* to prune this combinatorial search space, i.e. the support metric is used to separate infrequent subgraphs from the frequent ones.

Frequent subgraph mining plays an essential role in many graph mining applications such as chemical compound analysis [Huan et al., 2004c, Deshpande et al., 2005], document image clustering [Barbu et al., 2005], software bug isolation [Liu et al., 2005, Eichinger et al., 2008], web content mining [Schenker et al., 2004], social network mining [Mukherjee and Holder, 2004, Yang et al., 2006, Lahiri and Berger-Wolf, 2007], email mining [Aery and Chakravarthy, 2005a,b], and anomaly detection [Noble and Cook, 2003, Eberle and Holder, 2007]. Frequent subgraph mining is thus focus of the work described in this thesis.

1.1 Research Motivation

As noted above, the most common approaches to frequent subgraph mining adopt the support metric. However, use of the support metric gives rise to five significant disadvantages:

(a) **Computational complexity**: Frequent subgraph mining, using the support metric, has been demonstrated to work well in domains [Inokuchi et al., 2000, Kuramochi and Karypis, 2001, Huan et al., 2003, Borgelt and Berthold, 2002, Yan and Han, 2002, Nijssen and Kok, 2004, Deshpande et al., 2005] where the individual subgraphs are relatively small. However, when frequent subgraph mining is applied to more substantial domains, including image mining, text mining and social network mining, the computational complexity becomes very high due to the "combinatorial explosion" encountered with respect to the number of possible patterns. Many existing approaches to frequent subgraph mining cannot cope with large graph sets.

(b) **Large number of patterns generated:** Because a low support threshold is typically used to ensure that significant patterns are not missed, a large number of patterns are often generated. The resulting collection of patterns also typically includes significant amounts of repetition and/or redundancy. Furthermore, analysing large collections of patterns is both difficult and resource intensive.

(c) **Lack of control of the generation process:** Users can only control the number of patterns generated by adjusting this support threshold. A high support threshold will reduce the number of patterns detected, but at the risk of missing significant patterns (hence low support thresholds are typically used). Use of a low support threshold to ensure that all interesting patterns are discovered, however, entails a significant computational overhead.

(d) **Not all significant patterns may be identified:** For real applications, the support metric is often not adequate to catch the relative importance of all patterns; the significance of patterns is not necessarily encapsulated by a support count alone.

Alternative methods to reducing the search space include concentrating on the identification of a subset of the total set of frequent subgraphs, for example, closed frequent subgraph mining [Yan and Han, 2003] or maximal frequent subgraph mining [Huan et al., 2004b, Thomas et al., 2006]. Although these methods address the issue to some extent, the combinatorial explosion issue is still unresolved; closed frequent subgraph mining and maximal frequent subgraph mining still generate significantly large numbers of patterns especially on dense graph data sets [Bringmann and Zimmermann, 2009]. Consequently the graph mining result remains difficult to explore and interpret. With respect to identifying the most significant patterns one approach is to integrate some constraints into the frequent subgraph mining process [Yan et al., 2007, Zhu et al., 2007]. However, the constraints employed tend to be directed at basic graph properties (e.g. requiring the average density of a pattern to be over some threshold), the characteristics related to the properties of the vertexes and/or edges of the graph are typically not considered. There are, of course, also alternative interestingness measures to the support metric that may be employed. For example a variety of interestingness measures have been proposed in the context of association rule mining (e.g. [Tan et al., 2002]), some of which can be adopted for frequent subgraph mining. However, in many cases, a single metric is too general to define interestingness. Alternative solutions ([Xin et al., 2006c, Huan et al., 2004c, Yan et al., 2008]), that have mainly been applied within the chemical analysis domain, require some additional information to be provided by the user. There is little research work on adopting alternative generic interestingness measures to assess the importance of subgraph patterns.

To address the above this thesis proposes weighted frequent subgraph mining. The intuition here is that for many applications some vertexes and/or edges can be consid-

ered to be much more significant than others; consequently, by using weightings, the search space can be reduced in such a way that the most interesting patterns are still discovered, and none of the less interesting patterns. By integrating weight constraints into the work of frequent subgraph mining, it is expected that a smaller set of weighted frequent subgraphs can be discovered within a reasonable time-limit where established frequent subgraph mining algorithms without weightings can not achieve this. This intuition is not necessarily applicable to all frequent subgraph mining applications, but it is suggested in this thesis that there are many applications where this is the case. Examples include:

(a) Image classification

Frequent subgraph pattern based classification comprises two phases. Firstly, a frequent subgraph mining algorithm is used to extract frequent patterns from the graph database; secondly, the identified patterns are employed to build feature vectors which can be used as input to established classification algorithms (e.g. decision tree [Quinlan, 1993], CBA [Liu et al., 1998], and SVM [vapnik, 1999]). In this situation, high support values may engender the absence of important patterns and low support values may cause the problem of "memory overflow" and "high dimensionality" due to the nature of frequent subgraph mining and the effort required to construct feature vectors. In the context of graph represented images, two weak points are introduced, due to the nature of the diverse types of images.

- The size of the graphs used to represent images (in terms of the number of vertexes or edges) is substantial. During the frequent subgraph mining, the processing of graphs requires more runtime and more memory resource to store the intermediate results.
- It is difficult to identify distinct labels for vertexes or edges in the graph. Less distinct labels for vertexes or edges will result in a computational overhead.

These two points adversely affect the performance of frequent subgraph mining when applied to graph represented image sets. It is proposed in this thesis that weights may be attached to vertexes or edges that make up image graphs to indicate their "strengths". It is expected that the performance of image classification using extracted weighted frequent subgraphs can be achieved at a competitive level.

(b) Document categorization

The most common document formalisation for text classification is the *vector space model* (VSM) [Salton et al., 1975] founded on the bag of words/phrases representation. The main advantage of the vector space model is that it can readily be employed by classification algorithms (e.g. decision tree, naive Bayesian classifier). However, the bag of words/phrases representation is suited to capturing only

word/phrase frequency; structural and semantic information such as order and the proximity of word occurrence is ignored. It has been established that structural information plays an important role in classification accuracy [Deshpande et al., 2005]. An alternative to the bag of words/phrases representation is a graph based representation, which intuitively possesses much more expressive power. However, as noted above, this representation introduces an additional level of complexity in that the calculation of the similarity between two graphs is significantly more computationally expensive than between two vectors (e.g. [Schenker, 2003]). Some work [Markov and Last, 2005] has been done on hybrid representations to capture both structural elements (using the graph model) and significant features (using the vector model). However the computational resources required to process this hybrid model are still costly due to:

- The extremely high number of vertexes and edges, and low number of edge labels and high repetition of vertex labels.
- The consequent exponential complexity of the search space.

The computational complexity of the graph representation for document classification is the main disadvantage of the approach and prevents the full exploitation of the expressive power that the graph representation possesses. It is suggested that the work described in this thesis will address this issue through the application of the proposed weighted frequent subgraph mining. By combining the advantages of both the VSM and the graph representation, it is suggested that each vertex in the graph representation can be assigned a weight computed by the term weightings as used in information retrieval, and each edge can be assign a weight computed by some similarity measure (e.g. cosine similarity) between two vertexes. It is conjectured that weighted frequent subgraph mining when applied to such weighted graphs will generate fewer weighted frequent subgraphs in a smaller amount of runtime. More importantly, it is conjectured that the extracted weighted frequent subgraphs are the "right" patterns for document classification purpose.

(c) Network mining

For studies in domains such as communication networks, and social networks, the connections in the network are usually assumed to be binary valued (either present or absent). However, it is common in real-world networks [Barrat et al., 2004, Ebel et al., 2002, Kossinets and Watts, 2006] for connections to be weighted by assigning different strengths, intensities or capacities to these connections [Wasserman and Faust, 1994]. For instance, some social networks feature both strong and weak social connections between individuals; in transportation networks, it may be important to calculate the weight of the connections by considering the amount of traffic

flowing along them [Barrat et al., 2004]. By incorporating weights into the work of network mining, it is expected that these weights can be utilized to compute a significance factor for each discovered pattern. Thus applying established frequent subgraph mining to large networks becomes realisable by adopting weighted frequent subgraph mining; otherwise, such networks tend to be too large and complex for the application of established frequent subgraph mining algorithms. In addition, the extracted weighted frequent subgraphs can be employed to (say) cluster network vertexes or detect changes in the behaviour of the network.

1.2 Research Question

With respect to the foregoing, single support metric based frequent subgraph mining tends to be applicable only when the dataset to be mined is of moderate size and the graph structure is sparse. If the support value is relatively low, the efficiency of frequent subgraph mining algorithms becomes an issue [Han et al., 2007]. Furthermore, a high proportion of the discovered frequent subgraphs are often found to be repetitive and redundant in terms of their usefulness with respect to further analysis (e.g. classification) [Yan and Han, 2003, Huan et al., 2004a, Fan et al., 2008, Yan et al., 2008].

Existing frequent subgraph mining algorithms [Cook and Holder, 1994, 2000, Inokuchi et al., 2000, Kuramochi and Karypis, 2001, Huan et al., 2003, Yan and Han, 2002, Nijssen and Kok, 2004] tend to assume that all discovered frequent subgraphs have equal importance. It is suggested in this thesis that some subgraphs are more important than others according to significance factors that may be associated with them. Therefore, a *weighting* concept is proposed to assign non-negative real values to each discovered frequent subgraph to indicate their significance. It is conjectured that by integrating weights into the frequent subgraph mining process, a smaller and more significant subset of the complete set of frequent subgraphs may be discovered.

The research question posed in this thesis is thus: *Can vertex and/or edge weighing functions be usefully employed, in the context of frequent subgraph mining, so as to discover the most significant subgraphs (i.e. fewer subgraphs) in a manner that is more computationally efficient than established approaches to frequent subgraph mining?* There are a number of sub-questions associated with this research question:

(a) **What form should the desired weighting function take?** Weights may be applied to either vertexes or edges or both, the criteria for selecting which is the most appropriate is not clear, but is likely to be application dependent.

(b) **How should weightings be derived?** Weightings may be initially provided by end users or may be derived according to the structure of identified subgraphs. Which is the most desirable is in part a resource issue and in part also application dependent.

(c) **What is the nature of the data structures that would be required to support weighted frequent subgraph mining?** Weighted frequent subgraph mining necessitates operating on different kinds of graphs (e.g. undirected graphs, trees, or directed graphs). Thus the nature of the data structures to be employed in weighted frequent subgraph mining is important.

(d) **How should weightings be used?** For each identified subgraph how weightings can best (most effectively) be applied to determine the importance of a graph is a significant issue.

(e) **Whether to maintain the Downward Closure Property (DCP) or not.** The DCP [Agrawal et al., 1993, Pei et al., 2001a], which states that a graph can only be frequent if all of its subgraphs are also frequent, is generally used to reduce the computational overhead of mining. Thus, whether any devised weighting schemes should satisfy the DCP, or whether some alternative scheme obviating the need for the DCP may be derived, is of importance.

1.3 Methodology

To provide an answer to the proposed research question the broad research methodology that was adopted was to consider a sequence of frequent subgraph mining application scenarios. This was because it was felt that many of the research issues to be addressed were application dependent. A large number, ten in total, of such scenarios were considered; selected from a variety of different domains. An itemized list of the graph data considered is presented in Table 1.1 (Note that in some cases, data comprised several data sets.). In the table, 'ST' denotes synthetic tree data, 'RT' denotes real tree data, 'SG' denotes synthetic graph data, and 'RG' denotes real graph data. Note that the last eight are all real data.

To further facilitate the proposed study the frequent subgraph mining domain was first divided, according to the nature of the graphs to be processed, into three categories: (i) trees, (ii) undirected graphs, and (iii) directed graphs. Consequently, the weighted frequent subgraph mining investigation was split into weighted tree mining, weighted undirected graph mining, and weighted directed graph mining. In this context it is also worth noting that current frequent subgraph mining algorithms are typically applicable to undirected graphs only, although in real applications structured data can also often be represented as directed graphs. Therefore, using a major frequent subgraph mining algorithm, gSpan, as a base algorithm, a modified gSpan was implemented by the author to handle directed graphs (see Chapter 6 for details).

Having identified three categories of graph data the domain was further sub-divided according to the nature of the weighting functions that might be applied. Two categories of function were identified: (i) structural weighting which utilizes the structural

information associated with subgraphs, and (ii) content weighting which utilizes the domain user's knowledge. Each was considered with respect to the above graph data categorisation.

The research domain was thus divided into a two-by-three categorisation. The issue of whether to apply proposed weighting strategies so that the the DCP was maintained or to identify some alternative strategy that obviates the need for the DCP was considered with respect to each of the identified categories.

Table 1.1: Summary of graph data sets

Data	Description	# Datasets
ST1	Synthetic trees created by a random tree generator	2
ST2	Synthetic images represented by quad-trees (one tree per image)	3
RT1	Three collections of web logs (CSLOGS) represented by trees	3
RT2	A collection of MRI scanned brain images represented by quad-trees	3
RT3	A collection of Medline text documents represented by trees	1
RG1	Two collections of chemical compounds represented by undirected graphs	2
RG2	A collection of mammographic images represented by attributed relational graphs	2
RG3	A collection of real-world object images represented by attributed relational graphs	2
RG4	Three collections of text documents represented by directed graphs	3
RG5	A sample taken from the UK Cattle Movement database represented using directed graphs	3

1.4 Evaluation Criteria

The primary objective of the proposed research was to devise mechanisms to reduce the search space and the consequent number of discovered subgraphs by concentrating on the most "significant" subgraphs. The term "significant" in this context is of course subjective and difficult to measure. An evaluation mechanism was required that would demonstrate that the right subgraphs had been found. This was achieved by directing the work so that, for many of the scenarios, a classification scenario could be formulated. The objective of classification, within the context of data mining, is to build a classifier using pre-labelled training data, test this using pre-labelled test data and (assuming the results of the testing are satisfactory) apply the classifier to unseen data [Han and Kamber, 2006]. In the context of the work described in this thesis, the identified frequent subgraphs were used to define *feature vectors* which could then be input to established classification algorithms. For this purpose, labelled graph data sets

were used that could be divided into a training and a test set. Any proposed weighted frequent subgraph mining approach could then be applied to the training set and the resulting weighted frequent subgraphs used to define feature vectors which could be fed into a standard classifier generator and the accuracy of the resulting classifier ascertained using the test set. The operation of weighted frequent subgraph mining could then be compared to non-weighted frequent subgraph mining in terms of classification accuracy. If the accuracy of the weighted approach exceeded, or was at least comparable with, the non-weighted approach it could be argued that the correct subgraphs had been identified. This evaluation process is illustrated in Fig.1.2. The figure should be read from the top left to bottom right. The top part of the figure illustrates the process of modelling various types of data in terms of different kinds of graphs. The bottom part illustrates the process of building feature vectors from the extracted weighted frequent subgraphs and then feeding them into a classification algorithm.

The secondary objective of the proposed work was to devise weighted subgraph mining mechanisms that were efficient. The efficiency of the weighted frequent subgraph mining was considered in terms of runtime. Efficiency could also have been considered in terms of storage requirements, however it was found that it was not always possible to ascertain this, because for many of the considered scenarios the size of the graph data sets were such that the non-weighted approach exhausted memory resources.

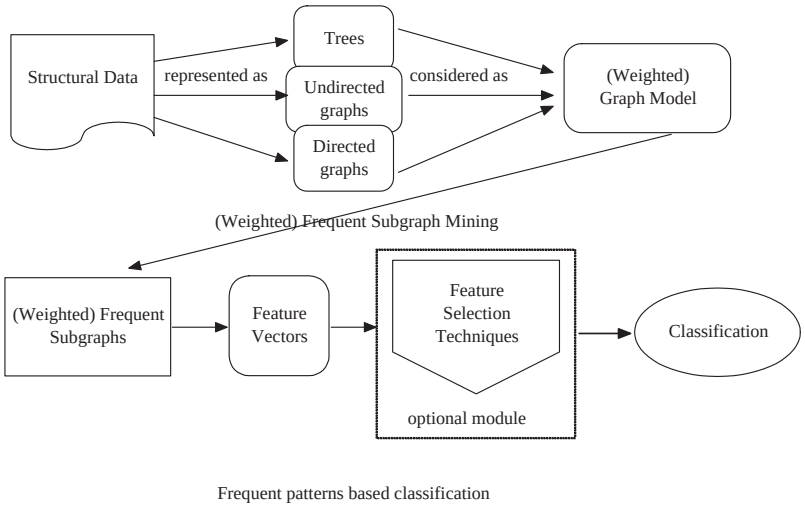


Figure 1.2: Evaluation model for weighted frequent subgraph mining

1.5 Contribution

The main contributions of the research work considered in this thesis can be summarized as follows:

1. The *weighting* concept, which puts an emphasis on the most important frequent

subgraphs instead of identifying all the frequent subgraphs during the mining process, was proposed (see Chapters 2, 4, 5, and 8).

2. The definition of a number of weighting functions to determine the weights (significance) of vertexes or edges in graphs was introduced (see Chapter 4).

3. A sequence of subgraph weighting schemes, to attach significance to identified subgraphs, which can be integrated seamlessly into the process of mining frequent subgraphs, was devised (see Chapter 5).

4. A number of weighted frequent subgraph mining algorithms, which founded on different weighting strategies and directed at different types of graphs, were devised (see Chapters 5 and 8).

5. A framework for integrating *feature selection* techniques into the weighted frequent subgraph mining process in the context of frequent pattern based graph classification (Figure 1.2) was introduced (see Section 5.2.3).

6. A mechanism for the domain user to control the mining process by adjusting either the support or the weighting threshold was introduced, such that the computational complexity of frequent subgraph mining is reduced in a trade-off between efficiency and effectiveness (see Chapters 5 and 8).

7. A new framework for image classification using an image interest points based graph representation in the context of the weighted frequent subgraph mining was proposed (see Section 3.2.5).

8. A systematic framework was proposed for classifying documents using a graph based representation (see Sections 3.2.3 and 3.2.7), in conjunction with weighted frequent subgraph mining (see Chapter 6, 7 and 8).

1.6 Organization of the Thesis

The rest of the thesis is organized as follows. In Chapter 2 the background (literature review) to the work described is introduced. The graph data sets used for evaluation purposes in this thesis are then described in Chapter 3. Weighting functions for both vertexes and edges are examined in Chapter 4. Algorithms integrating the various weighting functions into the frequent subgraph mining process and the experimental results are then considered in Chapters 5, 6, 7, and 8. Chapter 5 considers weighted frequent subgraph mining in the context of maintaining the DCP. Chapter 6 then presents the empirical results obtained by applying the algorithms introduced in Chapter 5 to the data sets introduced in Chapter 3. Chapter 7 investigates two case studies of applying the algorithms introduced in Chapter 5. Chapter 8 presents an alternative to

weighted frequent subgraph mining in the context of not maintaining the DCP, and reports the experimental result. Finally, some discussions and conclusions are presented in Chapters 9 and 10 respectively.

Chapter 2

Background

This chapter provides the background knowledge with respect to the research work described in this thesis. The chapter commences, Section 2.1, with the concepts employed in this thesis. Then in Section 2.2, the definitions of frequent subgraph mining and weighted frequent subgraph mining are presented. A discussion of graph isomorphism detection, the most significant research issues associated with frequent subgraph mining, is presented in Section 2.3; Section 2.4 then provides an overview of the process of frequent subgraph mining. "State of the art" reviews of current work on frequent subtree and subgraph mining are then presented in Sections 2.5 and 2.6 respectively. Some applications of frequent subgraph mining are presented in the following two sections. Frequent subgraph based classification and clustering are considered in Section 2.7, and social network mining using frequent subgraphs in Section 2.8. Other research work that is related to the proposed research described in this thesis is discussed in Section 2.9. Finally, all the work that has been described in this chapter is summarized in 2.10.

2.1 Concepts

This section presents the necessary definitions, commonly adopted in graph theory, that are required as a precursor to any discussion concerning frequent subgraph mining. Generally speaking, a *graph* is defined in terms of a set of vertexes (nodes) which are interconnected by a set of edges (links) [Gibbons, 1985].

Labelled Graph: A labelled graph can be represented as $G(V, E, L_V, L_E, \varphi_v, \varphi_e)$, where V is a set of vertexes, $E \subseteq V \times V$ is a set of edges; L_V and L_E are vertex and edge labels respectively; and φ_v and φ_e are the corresponding functions that define the mappings $V \rightarrow L_V$ and $E \rightarrow L_E$. A list of basic graph vocabulary is presented in Table 2.1.

Subgraph: Given two graphs $G_1(V_1, E_1, L_{V_1}, L_{E_1}, \varphi_{V_1}, \varphi_{E_1})$ and $G_2(V_2, E_2, L_{V_2}, L_{E_2}, \varphi_{V_2}, \varphi_{E_2})$,

Table 2.1: Basic vocabulary

Vocabulary	Description
Path	A sequence of vertexes which can be ordered such that two vertexes form an edge if and only if they are consecutive in the sequence [West, 2000].
The *length* of a path	The number of edges in a path.
Cycle	A path where the start and the end vertexes of the path are the same.
Self-cycle (loop)	An edge that connects a vertex to itself.
Multiple edges	Two or more edges connecting to the same two vertexes.
An *acyclic graph*	A graph that contains no cycles.
A *(dis)connected* graph	A graph is connected if it contains a path for every pair of vertexes, and disconnected otherwise [West, 2000].
A *complete* graph	A graph where each pair of vertexes is joined by an edge.
A *directed* graph	A graph where each edge of a graph describes an ordered pair of vertexes.
A *undirected* graph	A graph where each edge of a graph describes an unordered pair of vertexes.

G_1 is a subgraph of G_2, if G_1 satisfies the following conditions.

$$V_1 \subseteq V_2, \quad \forall v \in V_1, \varphi_{V_1}(v) = \varphi_{V_2}(v),$$

$$E_1 \subseteq E_2, \quad \forall (u,v) \in E_1, \varphi_{E_1}(u,v) = \varphi_{E_2}(u,v) \ .$$

G_2 is also called a supergraph of G_1. Furthermore, G_1 is an *induced subgraph* of G_2, if G_1 further satisfies the following condition (in addition to the above conditions) [Inokuchi et al., 2002, Huan et al., 2003]:

$$\forall u, v \in V_1, (u,v) \in E_1 \Leftrightarrow (u,v) \in E_2 \ .$$

The definition of the *induced subgraph* indicates that a subgraph G_1 of a graph G_2 is induced if for any pair of vertexes that appear in both G_1 and G_2, the edges between the vertexes must also be present in both G_1 and G_2. In other words, an induced subgraph is a subgraph with some constraints.

Free Tree: An undirected graph that is connected and acyclic [Chi et al., 2004a,c].

Labelled Unordered Tree: A labelled unordered tree (an unordered tree, for short) is a directed acyclic graph denoted as $T(V, \phi, E, v_r)$, where V is a vertex set of T; ϕ is a labelling function, such that $\forall v_i \in V, \phi(v_i) \rightarrow v_i$; $E \subseteq V \times V$ is an edge set of T; and v_r is a distinguished vertex called *root* of T. For $\forall v_i \in V$, there is a unique path $(v_r, v_1, v_2, \cdots, v_i)$ from the root v_r to v_i [Asai et al., 2002, 2003].

If a vertex v_i is on the path from the root to the vertex v_j, then v_i is an *ancestor* of v_j, and v_j is a *descendant* of v_i. For each edge $v_i, v_j \in E$, v_i is the *parent* of v_j, and v_j is a *child* of v_i. Vertexes that share the same parent are *siblings*. The *size*

of T is defined to be the number of vertexes in T. A vertex without any child is a *leaf* vertex; otherwise it is an *intermediate* vertex. The *rightmost path* of T is the path from the root vertex to the *rightmost leaf*. The *depth/level* of a vertex is the length of the path from the root to that vertex. The *degree* of a vertex is the number of edges incident to the vertex [West, 2000, Chi et al., 2004a,c, Tan et al., 2005a].

Labelled Ordered Tree: A labelled ordered tree[1] (an ordered tree, for short) is a labelled tree with a left-to-right ordering imposed among the children of each vertex [Asai et al., 2002, 2003, Chi et al., 2004a].

Preorder Traversal: A preorder traversal of a general tree is one form of depth-first traversal which is performed recursively as follows: visit the root first, and then do a preorder traversal of each of the subtree of the root one-by-one in the order given [Preiss, 1998].

Postorder Traversal: A postorder traversal of a general tree is one form of depth-first traversal which is performed recursively as follows: do a postorder traversal of each of the subtrees of the root one-by-one in the order given, and then visit the root [Preiss, 1998].

Bottom-up Subtree: Given a labelled tree $T(V, \phi, E, v_r)$ (ordered or unordered), $T'(V', \phi', E', v'_r)$ is a bottom-up subtree of T if and only if (i) $V' \subseteq V$; (ii) $E' \subseteq E$; (iii) the labelling of V' and E' in T is preserved in T'; (iv) $\forall v \in V$, if $v \in V'$ then all descendants of v must also be in V'; (v) if T is ordered, then the left-to-right ordering among the siblings in T should be preserved in T' [Chi et al., 2004a, Valiente, 2002].

Induced Subtree: Given a labelled tree $T(V, \phi, E, v_r)$ (free tree or unordered tree or ordered tree), $T'(V', \phi', E', v'_r)$ is an induced subtree of T, if and only if (i) $V' \subseteq V$; (ii) $E' \subseteq E$; (iii) the labelling of V' and E' in T is preserved in T'; (iv) if defined for ordered trees, the left-to-right ordering among the siblings in T' should be a sub-ordering of the corresponding vertexes in T [Chi et al., 2004a, Tan et al., 2006].

Embedded Subtree: Given a labelled tree $T(V, \phi, E, v_r)$, $T'(V', \phi', E', v'_r)$ is an embedded subtree of T, if and only if (i) $V' \subseteq V$; (ii)$\forall v \in V', \phi'(v) = \phi(v)$; and (iii) $\forall (u, v) \in E'$ such that u is the parent of v, u is an ancestor of v in T; (iv) if defined for ordered trees, $\forall (u, v) \in V', preorder(u) < preorder(v)$ in T' if and only if $preorder(u) < preorder(v)$ in T, where the preorder of a vertex is its index in the tree according to the preorder traversal [Chi et al., 2004a].

[1] A labelled ordered tree is also called a *rooted plane tree* in graph theory [West, 2000].

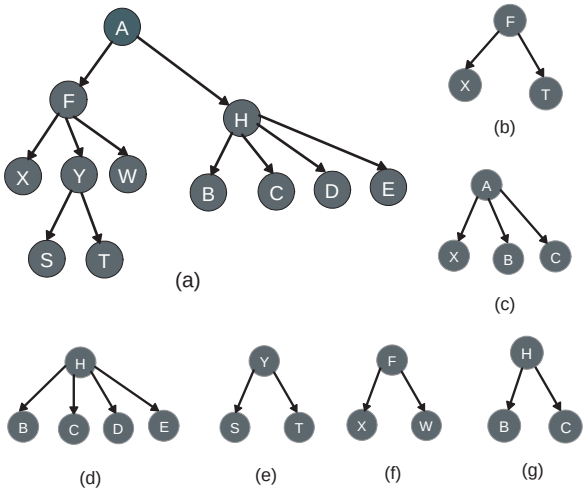


Figure 2.1: Different types of trees

Figure 2.1 shows examples of bottom-up subtrees, induced subtrees, and embedded subtrees. In Figure 2.1 tree (a) represents a given tree, trees (d) and (e) are two bottom-up subtrees of (a), trees (f) and (g) are two induced subtrees of (a), and trees (b) and (c) are two embedded subtrees of (a). The relationship among these three types of subtrees can be denoted as: {bottom-up subtrees} $\subseteq$ {induced subtrees} $\subseteq$ {embedded subtrees}.

Graph Isomorphism: A graph $G_1(V_1, E_1, L_{V_1}, L_{E_1}, \varphi_{V_1}, \varphi_{E_1})$ is isomorphic to another graph $G_2(V_2, E_2, L_{V_2}, L_{E_2}, \varphi_{V_2}, \varphi_{E_2})$, if and only if a bijection $f : V_1 \to V_2$ exists such that:

$$\forall u \in V_1, \quad \varphi_{V_1}(u) = \varphi_{V_2}(f(u)),$$

$$\forall u, v \in V_1, (u, v) \in E_1 \Leftrightarrow (f(u), f(v)) \in E_2,$$

$$\forall (u, v) \in E_1, \varphi_{E_1}(u, v) = \varphi_{E_2}(f(u), f(v)).$$

The bijection f is an isomorphism between G_1 and G_2. A graph G_1 is *subgraph isomorphic* to a graph G_2, denoted by $G_1 \subseteq_{sub} G_2$, if and only if there exists a subgraph g of G_2 such that G_1 is isomorphic to g [Huan et al., 2003]; g is called an *embedding* of G_1 in G_2.

Lattice ($\mathbb{GD}$): Given a database $\mathbb{GD}$, Lattice ($\mathbb{GD}$) is a structural form used to model the search space when finding frequent subgraphs, where each node represents a connected subgraph of the graphs in $\mathbb{GD}$. The lattice is typically depicted as shown in in Figure 2.2. The *lowest* node (bottom of the figure) represents the empty subgraph and the nodes at the *higher* levels represent all the graphs in $\mathbb{GD}$. A node p is a parent of a node q in the lattice, if q is a subgraph of p, and q is different from p by exactly one less edge. All the subgraphs of each graph

in $G_i \in \mathbb{GD}$ which occur in the database are present in the lattice and every subgraph occurs only once in it [Thomas et al., 2006].

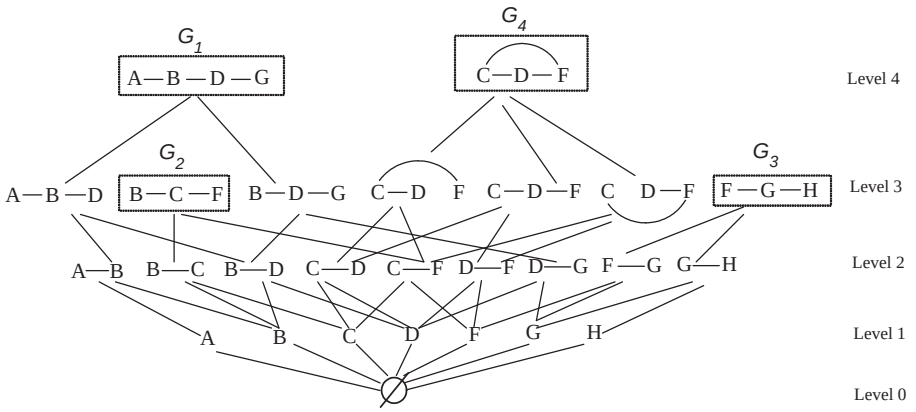


Figure 2.2: Lattice($\mathbb{GD}$)

Example: considering a graph database $\mathbb{GD} = \{G_1, G_2, G_3, G_4\}$, the corresponding *Lattice*($\mathbb{GD}$), which is in spirit similar to the example presented in Thomas et al. [2006], is given in Figure 2.2. In the figure, the lowest vertex ϕ represents the empty subgraph, and the vertexes at the highest level correspond to G_1, G_2, G_3, and G_4. The parents of the subgraph *B-D* are subgraphs *A-B-D* (joining the edge *A-B*) and *B-D-G* (joining the edge *D-G*). Similarly, subgraphs *B-C* and *C-F* are the children of the subgraph *B-C-F*.$\diamond$

2.2 Problem Definition

From the literature two separate problem formulations for Frequent Subgraph Mining (FSM) can be identified: (i) *transaction graph based*, and (ii) *single graph based*. In transaction graph based mining, the input data comprises a collection of relatively small simple graphs[2] (called *transactions* [Agrawal and Srikant, 1994]), whereas in single graph based mining the input data is a very large single graph. The research work described in this thesis focuses on transaction graph based mining. Table 2.2 lists the notation in relation to the transaction graph based mining, which will be employed through out this thesis.

In the context of transaction graph based mining, FSM aims to discover all the subgraphs whose occurrences in a graph database are over a user defined threshold. Formally, given a database $\mathbb{GD}$ comprised of a collection of graphs and a threshold $\sigma(0 < \sigma \leq 1)$, the occurrence of a subgraph g in $\mathbb{GD}$ is defined by $occ_{\mathbb{GD}}(g) = |\{G_i \in \mathbb{GD} | g \subseteq_{sub} G_i\}|$. Thus, the *support* of a graph g is defined as the fraction of the graphs

[2] A simple graph is an unweighted and undirected graph with no loops and no multiple edges between any two different vertexes [Gibbons, 1985, West, 2000].

Table 2.2: Notation used throughout the thesis

Notation	*Description*
$\mathbb{GD}$	A graph database.
G_i	A transaction graph such that $G_i \in \mathbb{GD}$.
k-(sub)graph	A (sub)graph of size k in terms of vertexes, or edges, or paths.
g_k	A k-(sub)graph.
C_k	A set of subgraph candidates of size k .
F_k	A set of frequent k-subgraphs.
$V(g)$	The vertex set of some graph g.
$E(g)$	The edge set of some graph g.
L_V	The set of all vertex labels in $\mathbb{GD}$.
L_E	The set of all edge labels in $\mathbb{GD}$.
$\lvert \cdot \rvert$	The cardinality of a set.

in $\mathbb{GD}$ to which g is subgraph isomorphic (the concept of subgraph isomorphism will be discussed in further detail later in this chapter):

$$sup_{\mathbb{GD}}(g) = occ_{\mathbb{GD}}(g)/|\mathbb{GD}| \tag{2.1}$$

A subgraph g is **frequent** if and only if $sup_{\mathbb{GD}}(g) \geq \sigma$. The **frequent subgraph mining problem** is to find all the frequent subgraphs in $\mathbb{GD}$. A frequent subgraph g is *closed* if none of its proper supergraphs have the same support that g has, and *maximal* if none of its proper supergraphs are frequent (see Sub-section 2.6.2.2 for formal definitions).

The general assumption in this thesis is that the graphs in $\mathbb{GD}$ can have weights associated with either their vertexes or their edges, and that these weights are real numbers that reflect the importance of the individual vertexes and edges in the database. The term *weighted subgraph* is adopted to indicate that the subgraph is weighted by some weighting function. The weighting can be derived automatically according to the characteristics of the graph data or be provided by the domain user. The problem of **weighted frequent subgraph mining** is to find all the frequent subgraphs in $\mathbb{GD}$ in such a way that the weightings are used to their best advantage.

2.3 Graph Isomorphism Detection

The central issue of concern in frequent subgraph mining is *graph isomorphism detection*, and by extension *subgraph isomorphism detection*. The significance is that (sub)graph isomorphism detection entails a considerable computational overhead and consequently much research effort has been directed at ways of reducing this overhead. In this section a review is presented of a number of the most significant (sub)graph isomorphism detection algorithms.

Generally, graph matching refers to finding a correspondence between the vertexes

and edges of two graphs that satisfies some constraints or optimality criteria such that similar structures are mapped to each other [Conte et al., 2004]. The matching process can be formulated in various ways such as: graph isomorphism detection [McKay, 1981], subgraph isomorphism detection [Ullmann, 1976], maximum common subgraph detection [McGregor, 1982], and graph edit distance computation [Sanfeliu and Fu, 1983]. Maximum common subgraph detection and graph edit computation are rarely used in frequent subgraph mining and consequently these will not be further discussed in the thesis (interested readers can refer to Bunke et al. [2002], Conte et al. [2004, 2007], Gao et al. [2010] for further details).

Graph isomorphism is a more stringent form of graph matching than subgraph isomorphism, i.e., a one-to-one correspondence must exist between the vertexes of one graph and the vertexes of the other so that adjacency is preserved [Fortin, 1996]. Graph isomorphism detection plays an essential role in determining the frequency of candidate subgraph patterns. Graph isomorphism is neither known to be solvable in polynomial time nor NP-complete[3] and subgraph isomorphism is known to be NP-complete [Garey and Johnson, 1979]. When restricting the graphs to trees, (sub)graph isomorphism detection becomes (sub)tree isomorphism detection. Tree isomorphism detection can be solved in a linear time Hopcroft and Tarjan [1972]. Faster subtree isomorphism detection algorithms with worst case time complexity of $\mathcal{O}(k^{1.5}n)$ were published in Matula [1978] and Chung [1987], and improved upon by Shamir and Tsur [1999] who claimed a worst case time complexity or $\mathcal{O}(\frac{k^{1.5}}{\log k}n)$ time (k and n are the sizes of the subtree and the tree to be searched in terms of the number of vertexes).

Subgraph isomorphism detection has been applied to a variety of domains, such as pattern recognition [Lu et al., 1991], shape analysis [Pearce et al., 1994], computer vision [Wong, 1992], and machine learning [Cook and Holder, 1994]. Subgraph isomorphism detection is fundamental to frequent subgraph mining. Many "efficient" frequent subgraph mining algorithms have been proposed directed at avoiding or reducing the operation of subgraph isomorphism detection. A significant number of subgraph isomorphism detection algorithms have been reported in the literature, which can be roughly categorized as being either exact matching [Ullmann, 1976, Schmidt and Druffel, 1976, McKay, 1981, Cordella et al., 1998, 2001] or error tolerant matching [Shapiro and Haralick, 1981, Bunke and Allerman, 1983, Christmas et al., 1995, Messmer and Bunke, 1998]. Most of the frequent subgraph mining algorithms use exact matching. An itemised overview of the main exact matching (sub)graph isomorphism detection algorithms is presented in Table 2.3. In Table 2.3, column two indicates the main methods employed to carry out the isomorphism detection, and column three indicates

[3]NP-complete: In computational complexity theory, the complexity class NP refers to the set of decision problems whose solutions can be verified in polynomial time. A decision problem ρ is NP-complete if ρ is in NP and every other problem in NP is reducible to ρ in polynomial time [Cormen et al., 2001].

whether the isomorphism detection algorithm is applicable to graph isomorphism, subgraph isomorphism or both.

Table 2.3: Overview of a number of exact matching (sub)graph isomorphism algorithms

Algorithms	Main Techniques	Matching Types	Time Complexity (Worst Case)
Ullmann	backtracking look ahead function	graph & subgraph isomorphism	$\mathcal{O}(\|V(g)\|!\|V(g)\|^3)$
SD	distance matrix backtracking	graph isomorphism	$\mathcal{O}(\|V(g)\|!\|V(g)\|)$
Nauty	group theory canonical labelling	graph isomorphism	exponential
VF	DFS strategy feasibility rules	graph & subgraph isomorphism	$\mathcal{O}(\|V(g)\|!\|V(g)\|)$
VF2	VF's rationale advanced data structures	graph & subgraph isomorphism	$\mathcal{O}(\|V(g)\|!\|V(g)\|)$

With respect to Table 2.3, Ullmann's algorithm [Ullmann, 1976] is a widely used graph matching algorithm [Messmer, 1996]. The algorithm employs a backtracking procedure with a look-ahead function to reduce the size of the search space. Similarly, the SD algorithm [Schmidt and Druffel, 1976] utilizes the distance matrix representation of a graph together with a backtracking procedure to reduce the search. McKay's Nauty algorithm [McKay, 1981] uses group theory to transform the graphs to be matched into a canonical form that allows for more effective graph isomorphism testing. However, the construction of the canonical form can lead to exponential complexity given a worst case scenario [Conte et al., 2004]. Although Nauty was regarded as the fastest graph isomorphism algorithm by Conte et al. [2004], Miyazaki [1997] demonstrated the existence of some categories of graph which required exponential time to generate the canonical labelling. The VF [Cordella et al., 1998] and VF2 [Cordella et al., 2001] algorithms use a depth first search (DFS) strategy, assisted by a set of feasibility rules to prune the search tree. VF2 is an improved version of VF that explores the search space more effectively so that the matching time and the memory consumption are significantly reduced.

In Foggia et al. [2001] a detailed experimental analysis of these five algorithms (Ullmann, SD, Nauty, VF and VF2) is provided to indicate that none of the existing algorithms is completely superior to the others. In general, VF2 was found to give the best performance with respect to the size and the type of graphs to be matched. In the framework of frequent pattern based classification described in Figure 2.9, VF2 algorithm was adopted in this thesis for use as the subgraph isomorphism detection algorithm to build feature vectors from the weighted frequent subgraphs discovered by the weighted frequent subgraph mining algorithm.

2.4 Overview of Frequent Subgraph Mining

As noted above a central theme of graph mining is frequent subgraph mining, which has demonstrated its advantages in various tasks such as chemical compound classification [Deshpande et al., 2005, Huan et al., 2004c], document image clustering [Barbu et al., 2005], graph indexing [Shasha et al., 2002, Yan et al., 2004], graph searching [Yan et al., 2005b, 2006a, Chen et al., 2007a], and many others. A generic overview of the process of frequent subgraph mining is presented in this section.

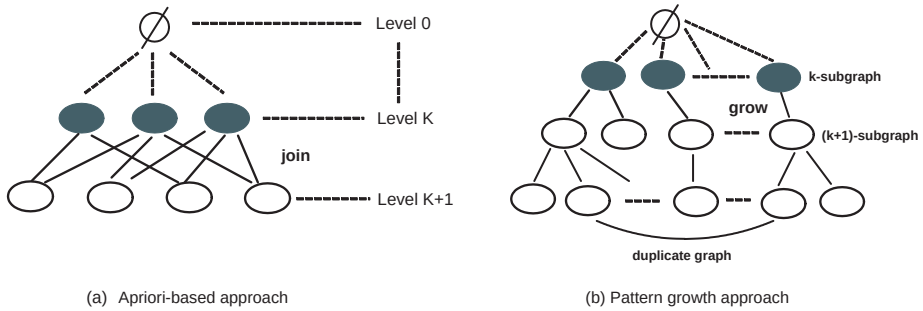


Figure 2.3: Two types of FSM approaches. Note that the subgraph lattice is shown "upside-down". Vertexes corresponding to graphs with fewer edges are displayed at the top of the picture in each case.

Algorithm 2.1: Apriori-based approach

Input: $\mathbb{GD}$ = a graph dataset, σ= minimum support
Output: $\mathcal{F}_1, \mathcal{F}_2, \cdots, \mathcal{F}_k$, a set of frequent subgraph sets

```
1  F_1 ← detect all frequent 1-subgraphs in GD
2  k ← 2
3  while F_{k-1} ≠ ∅ do
4      F_k ← ∅
5      C_k ← candidate-gen(F_{k-1})
6      foreach candidate g_k ∈ C_k do
7          g_k.count ← 0
8          foreach G_i ∈ GD do
9              if subgraph-isomorphism(g_k, G_i) then
10                 g_k.count ← g_k.count + 1
11             end
12         end
13         if g_k.count ≥ σ|GD| ∧ g_k ∉ F_k then
14             F_k = F_k ∪ g_k
15         end
16     end
17     k ← k + 1
18 end
```

It is widely accepted that FSM techniques can be divided into two categories: (i)

Algorithm 2.2: Pattern growth approach

Input: g = a frequent subgraph, σ = minimum support, $\mathbb{GD}$ = a graph dataset
Output: $\mathcal{F}$, a set of frequent subgraphs

```
1  F ← ∅
2  F_1 ← detect all frequent 1-subgraphs in GD
3  k ← 1
4  foreach g ∈ F_1 do
5  |  Pattern-growth(g, GD, σ, F)
6  end
7  Function: Pattern-growth(g, GD, σ, F)
8  k ← k + 1
9  C_k ← ∅
10 if g ∈ F then
11 |  return
12 else
13 |  F ← F ∪ g
14 end
15 scan GD, find all the edges e such that g can be extended to g ∪ e, g ← g ∪ e,
   and insert g into C_k
16 foreach g_k ∈ C_k do
17 |  if g_k.count ≥ σ|GD| then
18 |  |  Pattern-growth(g_k, GD, σ, F)
19 |  else
20 |  |  return
21 |  end
22 end
```

the Apriori-based approach (also called the BFS strategy based approach) and (ii) the pattern growth approach. These two categories are similar in spirit to counterparts found in association rule mining (ARM), namely the Apriori algorithm [Agrawal and Srikant, 1994] and the frequent pattern growth (FP-growth) algorithm [Han et al., 2000] respectively. As illustrated in Figure 2.3(a), the Apriori-based approach proceeds in a "generate-and-test" manner using a breadth first search (BFS) strategy to explore the subgraph lattice of the given database. Therefore, before exploring any $(k+1)$-subgraphs, all the k-subgraphs should first be explored. Further, any $(k+1)$-subgraph candidates are generated by joining two frequent k-subgraphs. Because there are many ways to join two frequent k-subgraphs, the Apriori-based approach may incur high complexity. The basic Apriori-based process is outlined in Algorithm 2.1. A pattern growth approach is shown in Figure 2.3(b). It can use both BFS and DFS strategies, but the latter is preferable to the former because it requires less memory usage. As illustrated in Figure 2.3, the difference between the pattern growth approach using a BFS strategy and the Apriori-based approach is that any $(k+1)$-subgraph candidates

are generated by joining two frequent k-subgraphs for the latter while for the former, any $(k+1)$-subgraph candidates are generated by extending one frequent k-subgraph. It can be observed in Figure 2.3 (b) that the same subgraph can be discovered many times. Theoretically, the same k-subgraph may be generated by extending one edge from k different $(k-1)$-subgraphs. Thus, how to extend a subgraph efficiently in order to reduce the generation of duplicate subgraphs is vital for a pattern growth approach. The basic pattern growth procedure, following the description presented in Han and Kamber [2006], is outline in Algorithm 2.2. As can be seen in Algorithm 2.2, for each discovered frequent subgraph g, this approach grows g recursively until all the frequent supergraphs of g are discovered.

According to the *downward closure property (DCP)* of frequent item sets (also called Anti-monotonicity property), as advocated in Association Rule Mining, if a graph is frequent then all of its subgraphs are also frequent. Therefore all frequent k-subgraphs are used to generate the $(k+1)$-subgraph candidates. If any of the candidate $(k+1)$-subgraphs are then found to be not frequent, they can be pruned. As exhibited in Algorithms 2.1 and 2.2, most existing frequent subgraph mining algorithms adopted an iterative pattern mining strategy. Each iteration can typically be divided into two clear phases: (i) candidate generation (line 5 in Algorithm 2.1 and line 15 in Algorithm 2.2) and (ii) support count computation (lines 6-12 in Algorithm 2.1 and line 15 in Algorithm 2.2). For the former, graph isomorphism detection is required in order to reduce the generation of duplicate subgraphs. The latter requires subgraph isomorphism detection. Generally, research on frequent subgraph mining focuses on these two phases. Since it is harder to address subgraph isomorphism detection, more research effort has been directed at how to efficiently generate subgraph candidates. Because subtree isomorphism detection can be solved in $\mathcal{O}(\frac{k^{1.5}}{\log k}n)$ time, the computational complexity of FSM is reduced within the context of trees. Therefore, in this thesis a distinction is made between frequent subgraph mining and frequent subtree mining. In the rest of this thesis the acronym FSM will be used to indicate both frequent subgraph and subtree mining; and the acronyms FGM and FTM will be used to distinguish between frequent subgraph and subtree mining respectively.

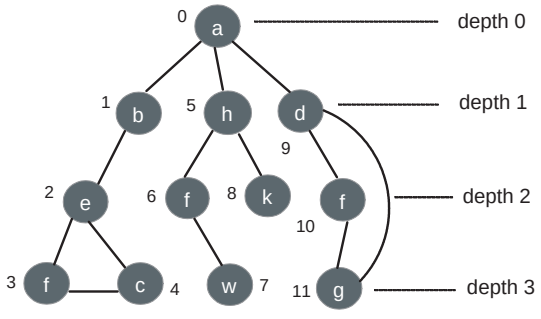
Before examining current FGM and FTM algorithms, how these algorithms represent and order the graphs and trees that they operate with will be considered first in Section 2.4.1.

2.4.1 Canonical representations

The simplest mechanism whereby a graph structure can be represented is by employing an *adjacency matrix* or *adjacency list*. The adjacency list is mainly used for the storage purposes in FSM when the graphs are sparse, while the adjacency matrix is mostly used for the canonical representation. Using an adjacency matrix the i-th row

and the i-th column of the matrix, (i, i), denotes the i-th vertex v_i, and the i-th row and the j-th column of the matrix, (i, j), denotes the potential edge connecting the vertexes v_i and v_j ($(i, j) = 0$ where no edges exist between v_i and v_j) [Inokuchi et al., 2003]. An example is given in Figures 2.4, where (b) is the adjacency matrix extracted from the graph presented in (a). In the figure, for ease of illustration, all edge labels are assumed to be the same and represented by '1'. Note that in Figures 2.4(a) the vertexes are numbered, in a depth first manner, from 0 to 11. The use of adjacency matrices, although straightforward, does not lend itself to isomorphism detection, because the vertexes (and edges) can be enumerated in many different ways [Washio and Motoda, 2003]. With respect to isomorphism testing, it is therefore desirable to adopt a consistent labelling strategy that ensures that any two identical graphs are labelled in the same way regardless of the order in which vertexes and edges are presented (i.e. a *canonical* labelling strategy).

A canonical labelling strategy defines a unique code for a given graph [Read and Corneil, 1977, Fortin, 1996]. Canonical labelling facilitates isomorphism checking because it ensures that if a pair of graphs are isomorphic, then their canonical labellings will be identical [Kuramochi and Karypis, 2001]. One simple way of generating a canonical labelling is to flatten the associated adjacency matrix by concatenating rows or columns to produce a code comprising a list of integers with a lexicographical ordering imposed. To further reduce the computation resulting from the permutations of the matrix, canonical labellings are usually compressed, using what is known as a *vertex invariant scheme* [Read and Corneil, 1977], that allows the content of an adjacency matrix to be partitioned according to the vertex labels. Various canonical labelling schemes have been proposed in the literature. Some of the most significant are briefly described below.



(a) Graph G with preorder subscripts

a	1	0	1	0	0	0	1	0	0
1	b	0	0	1	0	0	0	0	0
0	0	c	0	1	1	0	0	0	0
1	0	0	d	0	1	1	0	0	0
0	1	1	0	e	1	0	0	0	0
0	0	1	1	1	f	1	1	0	1
0	0	0	1	0	1	g	0	0	0
1	0	0	0	0	1	0	h	1	0
0	0	0	0	0	0	0	1	k	0
0	0	0	0	0	1	0	0	0	w

(b) G's adjacency matrix

Figure 2.4: A graph example with its adjacency matrix

Minimum DFS Code (M-DFSC): There are a number of variants of the Depth First Search (DFS) code canonical labelling scheme; but essentially each vertex is

given a unique identifier generated from a DFS traversal of the graph (DFS subscripting). Each constituent edge of the graph in the DFS code is then represented by a 5-tuple: (i, j, l_i, l_e, l_j), where i and j are the vertex identifiers, l_i and l_j are the labels for the corresponding vertexes, and l_e is the label for the edge connecting the vertexes. Based on the DFS lexicographic order, the M-DFSC of a graph g is defined as the canonical labelling of g [Yan and Han, 2002]. The DFS codes for the left-most branch and the right-most branch of the example graph G given in Figure 2.4(a) are $\{(0, 1, a, 1, b), (1, 2, b, 1, e), (2, 3, e, 1, f), (3, 4, f, 1, c), (4, 2, c, 1, e)\}$ and $\{(0, 9, a, 1, d), (9, 10, d, 1, f), (10, 11, f, 1, g), (11, 9, g, 1, d)\}$ respectively.

Canonical Adjacency Matrix (CAM): Given an adjacency matrix M of a graph g, an encoding of M can be obtained by the sequence of concatenating lower (or upper) triangular entries of M, including entries on the diagonal. Since different permutations of the set of vertexes correspond to different adjacency matrices, the canonical (CAM) form of g is defined as the maximal (or minimal) encoding. The adjacency matrix from which the canonical form is generated defines the Canonical Adjacency Matrix or CAM [Inokuchi et al., 2000, 2002, Kuramochi and Karypis, 2001, Huan et al., 2003]. The encoding for the example graph G given in Figure 2.4(a), represented by the matrix in Figure 2.4(b), is thus $\{a1b00c100d0110e00111f000101g1000010h00000001k000001000w\}$.

The above two schemes are applicable to any simple graph. It is easier to define a canonical labelling for trees than graphs because trees have an inherent structure associated with them. There also exist more specific labelling schemes that focus exclusively on trees. Among these, DFS-LS, DLS and CPS are directed at rooted ordered trees; whilst BFCS and DFCS are used for rooted unordered trees. Each is briefly described below where the tree example given in Figure 2.6 is used to illustrate the labelling schemes. In the figure, for ease of illustration, all edge labels are assumed to be the same and represented by '1'.

Canonical Representation of Free Trees: Free trees do not have roots. In this case a unique representation for a free tree is usually constructed by selecting one vertex or a pair of vertexes as the root(s). The procedure starts with removing all leaf vertexes and their incident edges recursively until a single vertex or two adjacent vertexes are left. In the first case, the remaining vertex is called the centre, and a rooted unordered tree is obtained with the centre as the root. The procedure is displayed in Figure 2.5(a). In the second case, the pair of remaining vertexes are called the bi-centre; a pair of rooted unordered trees can then be obtained with the bi-centre as the roots (along with an edge connecting two roots). The procedure is displayed in Figure 2.5(b). This pair of trees are ordered so that the root of the smaller one is chosen as the root of the whole tree

[Chi et al., 2003a, Ruckert and Kramer, 2004]. After obtaining rooted unordered trees, any canonical representations for rooted unordered trees can be employed to represent the free trees.

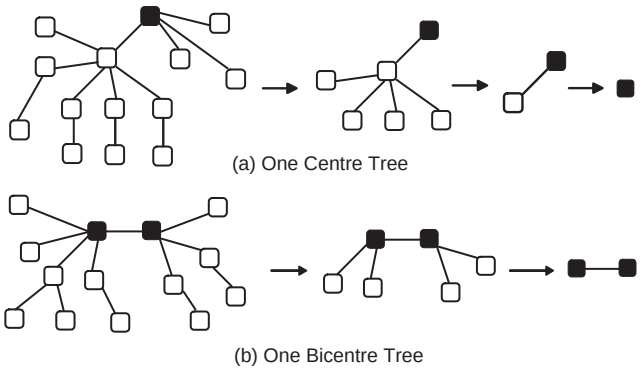


Figure 2.5: An example of two types of free trees

DFS Label Sequence (DFS-LS): Given a labelled ordered tree T, the labels of $\forall v_i \in V$ are added to a string S, during a DFS traversal of T. Whenever a backtrack occurs, a unique symbol "-1" or "\$" or "/" is added to S [Zaki, 2002, 2005b, Tan et al., 2006]. The DFS-LS code for the example tree T given in Figure 2.6 is $\{abea\$\$\$cfb\$d\$\$a\$\$dfc\$\$\$\}$.

Depth-Label Sequence (DLS): Given a labelled ordered tree T, depth-label pairs comprising the depth and the label of $\forall v_i \in V$, $(d(v_i), l(v_i))$, are added to a string S, during a DFS traversal of T. The depth-label sequence of T is defined as $S = \{(d(v_1), l(v_1)), \cdots, (d(v_k), l(v_k))\}$ [Asai et al., 2002, Nakano, 2002, Wang et al., 2004a]. DLS's variants can be further found in Asai et al. [2003] and Nijssen and Kok [2003]. The DLS code for the example tree T given in Figure 2.6 is $\{(0,a),(1,b),(2,e),(3,a),(1,c),(2,f),(3,b),(3,d),(2,a),(1,d),(2,f),(3,c)\}$.

Consolidated Prüfer Sequence(CPS): Given a labelled tree, the CPS encoding scheme consists of two parts: NPS that denotes an *extended prüfer sequence*[4] constructed using the postorder traversal numbers of vertexes as the set of unique labels; and LS that denotes a sequence of labels of deleted leaf vertexes at each step of a postorder traversal. Both the NPS and LS sequences jointly encode a unique representation for a labelled tree [Tatikonda et al., 2006]. The NPS and LS for the example tree T given in Figure 2.6 are $\{ebaffccafda-\}$ and $\{aebbdfaccfda\}$ respectively.

[4] An extended prüfer sequence (introduced by Tatikonda et al. [2006]) for a tree with n vertexes is a n-length sequence constructed by a recursive process with n iterations. At each iteration, the leaf with the smallest label (i.e. the smallest postorder traversal number) is removed and its parent is added to the already built partial prüfer sequence. When the last vertex is removed the corresponding entry in the sequence is denoted by "$-$".

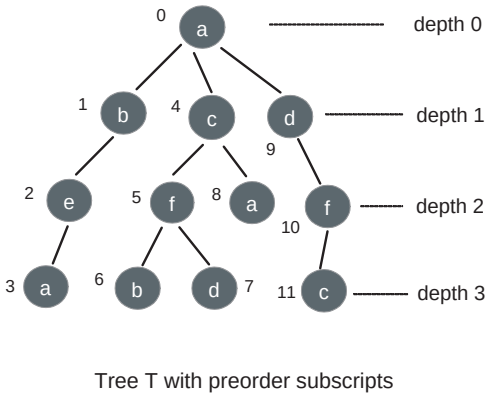


Figure 2.6: A tree example

Breadth-First Canonical String (BFCS): For a labelled ordered tree, every vertex label is added into a string, by traversing the tree in a BFS manner. Additionally, a "$" symbol is used to partition the families of siblings, and a "#" symbol to indicate the end of the string encoding. The "$" symbol is considered to be lexicographically before the symbol "#" and both are considered to be lexicographically after any other vertex and edge labels. Given an unordered tree T, different ordered trees with corresponding BFS string encodings can be produced by imposing different orders on the children of the intermediate vertexes. The BFCS of T is the lexicographically minimal of these encodings, and the corresponding rooted ordered tree defines the breadth-first canonical form (BFCF) of T [Chi et al., 2005]. BFCS's variants can be found in Chi et al. [2003a, 2004c]. Thus, a BFS string encoding of the example tree T given in Figure 2.6 is $a\$bcd\$e\$fa\$f\$a\$bd\$\$c\#$.

Depth-First Canonical String (DFCS): Similar to the BFCS but using DFS. The depth-first string encoding, for a labelled ordered tree, labels each vertex by traversing the tree in a DFS manner. The "$" symbol is used to represent a backtrack, and the "#" symbol to represent the end of string encoding. The DFCS of an unordered tree T is then the minimal of all the possible DFS encodings, according to the lexicographical ordering. The corresponding rooted ordered tree defines the depth-first canonical form (DFCF) of T [Chi et al., 2005]. A DFCS variant can be found in Chi et al. [2003a, 2004b]. A DFS string encoding of the example tree T given in Figure 2.6 is $abea\$\$\$cfb\$d\$\$a\$\$dfc\$\$\$\#$.

2.4.2 Frequency measures

In FSM, a subgraph candidate is said to be *frequent* if its frequency of occurrence is greater than or equal to a user defined threshold. The frequency of a pattern is calculated in terms of the *support* of the pattern. However, the manner in which

the support of a pattern is counted is dependent on the nature of the FSM problem formulation: either transaction graph based FSM (the focus of this thesis) or single graph based FSM.

In the context of transaction graph FSM, the support of a pattern is computed using *transaction based counting*; whereby the support for a candidate subgraph is determined according to the number of transaction graphs that the pattern occurs in, regardless of how many times the pattern actually occurs in a particular transaction graph. The support measure using transaction based counting can be formulated by Equation 2.1, which was introduced in Section 2.2. In the context of single graph based mining, the support of a pattern is computed by *occurrence based counting* whereby the support of a candidate subgraph is determined according to the total number of occurrences of a pattern in the input graph. Occurrence based counting may also be applied in the case of transaction graph FSM, but is more usually applied to single graph FSM (transaction based counting is clearly only applicable to transaction graph FSM). Transaction based counting offers the advantage that it satisfies the *DCP*, which can be employed to significantly reduce the computational overhead associated with candidate generation in FSM. Occurrence based counting does not feature the *DCP* (e.g. [Tan et al., 2006]): consequently an alternative measure, embracing occurrence based counting, which keeps the *DCP*, is required; or some heuristics are needed to limit the search space. There are a variety of support measures [Vanetik et al., 2002, Kuramochi and Karypis, 2004c, 2005, Vanetik et al., 2006] that have been proposed for single graph based FSM, which will be discussed in Sub-section 2.6.1.2. Since the focus of this thesis is on transaction graph based FSM, the further discussion of single graph based FSM is only included for completeness.

2.4.3 Candidate generation

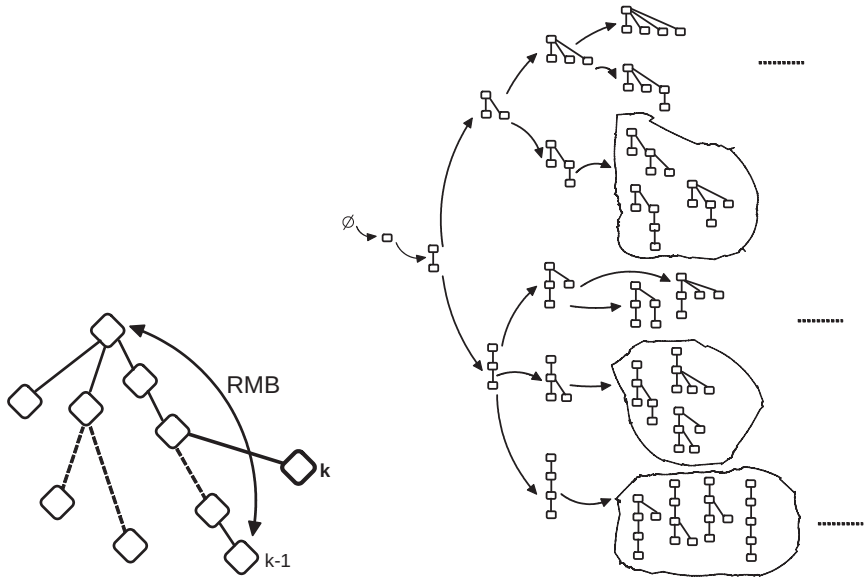
FSM can be conceptualised as a search through the lattice describing all possible patterns. One of two primary operations in FSM is candidate generation. For candidate generation the challenge is to systematically generate candidate subgraphs without redundancy (i.e. each subgraph should be generated at most once). Many different FSM algorithms have been characterized by the candidate generation strategy that they adopt. Some of the most significant will be briefly described below. A significant proportion of techniques employed in FTM are also applicable to FGM (and vice-versa), it is therefore difficult to distinguish between candidate generation techniques according to whether they are applicable to FTM or FGM.

2.4.3.1 Rightmost path expansion

Rightmost path expansion is the most common candidate generation strategy, which generates $(k + 1)$-subtrees from frequent k-subtrees by adding vertexes only to the

rightmost path of the tree [Asai et al., 2002, Zaki, 2002, Asai et al., 2003, Nijssen and Kok, 2003]. In Figure 2.7 (a), "RMB" denotes the rightmost branch of T, which is the path from the root to the rightmost leaf $(k-1)$; a new vertex k is added by attaching it to any vertexes along the RMB. An enumeration DAG (directed acyclic graph) using rightmost path expansion is a tree with a root ϕ, where each node is a subtree pattern. A node s is linked by another node t if and only if t is a rightmost path expansion of s. Every 1-subtree is a rightmost path expansion of the root ϕ and every $(k+1)$-subtree is a rightmost path expansion of the k-subtree. Hence, all subtree patterns can be enumerated by traversing in either a BFS or DFS manner [Asai et al., 2002].

Figure 2.7 (b) shows part of an enumeration DAG grown by rightmost path expansion. Each square in the figure represents a vertex in the tree. Each highlighted part represents all 5-subtrees obtained by the rightmost path expansion of the corresponding 4-subtrees. An enumeration DAG (sometimes also simplified as an *enumeration tree*) is used to illustrate how a set of patterns is completely enumerated in a search problem. Enumeration DAGs have been used extensively in ARM [Bayardo Jr., 1998, Agarwal et al., 2001], and subsequently, in a variety of ways, by many subtree mining algorithms [Asai et al., 2002, 2003, Nijssen and Kok, 2003, Chi et al., 2004c, 2005].



(a) The rightmost path (b) A partial enumeration DAG

Figure 2.7: An illustration of rightmost path expansion

2.4.3.2 Equivalence class based extension

The strategy of equivalence class based extension [Zaki, 2002, 2005b] essentially uses a DFS-LS representation for trees. Basically, a $(k+1)$-subtree is generated by joining two

frequent k-subtrees which must be in the same *equivalence class* $[C]$[5]. An equivalence class consists of the class prefix encoding, and a list of members. Each member of the class can be represented as a (l, p) pair, where l is the k-th vertex label and p is the depth-first position of the k-th vertex's parent. It is verified by Zaki [2002] that all potential $(k+1)$-subtrees with the prefix $[C]$ of size $(k-1)$ can be generated by joining each pair of members of the same equivalent class $[C]$.

2.4.3.3 Right-and-left tree join

The right-and-left tree join strategy was proposed by Hido and Kawano [2005]. It essentially uses the rightmost leaf (see 2.1) and leftmost leaf[6] of the tree to generate candidates in a BFS manner. Let $lml(t)$ denote the leftmost leaf of t and $Right(t)$ the right tree obtained by removing $lml(t)$; and let $rml(t)$ denote the rightmost leaf and $Left(t)$ the left tree obtained by removing $rml(t)$. Given two trees s and t where $Right(s) = Left(t)$, their right-and-left tree join is defined as: $join(s,t) = s \cup rml(t) = lml(s) \cup t$. A diagram depicting this join operation is displayed in Figure 2.8.

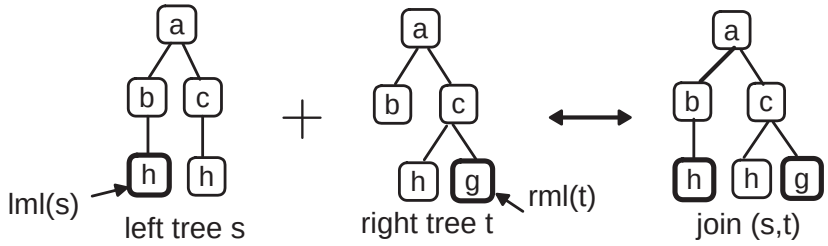


Figure 2.8: An illustration of right-and-left tree join

2.4.3.4 Extension and join

The extension and join strategy was first proposed by Huan et al. [2003], and later used by [Chi et al., 2004c]. It employed a BFCS representation such that a leaf at the bottom level of a BFCF tree is defined as a leg. For a node V_n in an enumeration tree, if the height of the BFCF tree corresponding to V_n is assumed to be h, all children of V_n can be obtained by either of the two following operations:

(a) Extension operation - adding a new leg at the bottom level of the BFCF tree yields a new BFCF with height $h+1$.

(b) Join operation - joining V_n and one of its sibling yields a new BFCF with height h.

[5]In Zaki [2002], two k-subtrees T_1, T_2 are in the same prefix equivalence class if and only if they share the same encoding up to the $(k-1)$-th vertex.

[6]The leftmost leaf of the tree, is the first leaf vertex in the DFS traversal of that tree [Hido and Kawano, 2005].

2.4.3.5 Level-wise join

The level-wise join strategy was introduced by Kuramochi and Karypis [2001]. Basically, a $(k+1)$-subgraph candidate is generated by joining two frequent k-subgraphs which have to share the same $(k-1)$-subgraph. This common $(k-1)$-subgraph is referred to as a *core* for these two frequent k-subgraphs. One main issue about this strategy is that one k-subgraph can have at most k different $(k-1)$-subgraphs and the joining operation may generate many redundant candidates. In Kuramochi and Karypis [2004b], this issue was addressed by limiting the $(k-1)$-subgraphs to be only two $(k-1)$-subgraphs with the smallest and the second smallest canonical labels. By carrying out this adapted join operation, the number of duplicate candidates generated was significantly reduced. Other algorithms adopting the strategy and its variants are AGM [Inokuchi et al., 2000], DPMine [Vanetik et al., 2002, Gudes et al., 2006], and HSIGRAM [Kuramochi and Karypis, 2004c], which will be discussed later.

Among these candidate generation strategies, the last two are centred on FGM and all others are concentrated on FTM.

Table 2.4: Taxonomy of frequent subtree mining algorithms

	Maximal	*Closed*	*Induced*	*Embedded*	T_c	O_c
Rooted unordered tree mining						
TreeFinder	⋆			⋆	⋆	
uFreqT			⋆		⋆	
Unot			⋆		⋆	
PathJoin	⋆		⋆		⋆	
cousinPair				⋆	⋆	
RootedTreeMiner			⋆		⋆	
SLEUTH				⋆	⋆	
Rooted ordered tree mining						
FREQT			⋆		⋆	
TreeMiner				⋆	⋆	
Chopper				⋆	⋆	
XSpanner				⋆	⋆	
AMIOT			⋆		⋆	
IMB3-Miner			⋆	⋆		⋆
TRIPS			⋆		⋆	
TIDES			⋆		⋆	
Free tree mining						
FreeTreeMiner			⋆		⋆	
FTMiner			⋆		⋆	
F3TM			⋆		⋆	
CFFTree		⋆	⋆		⋆	
Hybrid tree mining						
CMTreeMiner	⋆	⋆	⋆		⋆	
HybridTreeMiner			⋆		⋆	

2.5 Frequent Subtree Mining Algorithms

The previous section considered the issues of representation (canonical forms), frequency and candidate generation, in terms of both trees and graphs. In this section a number of prominent FTM algorithms are reviewed. FTM has attracted a great deal of research interest in areas such as: network IP multi-cast[7] routing [Cui et al., 2005], web usage mining [Cooley et al., 1997, Zaki, 2005a], XML mining [Zaki and Aggarwal, 2003, Tan et al., 2005b], bioinformatics [Hein et al., 1996, Ruckert and Kramer, 2004, Zhang and Wang, 2006], database indexing [Yang et al., 2003], computer vision [Liu and Geiger, 1999] and so on. The attraction of FTM is that the subgraph isomorphism detection problem becomes the subtree isomorphism detection problem, which can be solved in $\mathcal{O}(\frac{k^{1.5}}{\log k}n)$ time [Shamir and Tsur, 1999]. In addition the structure of trees may be usefully employed to simplify the overall mining process.

The FTM algorithms discussed in this section have been categorized, as shown in Table 2.4, according to the nature of the trees that they are directed at: (i) unordered trees, (ii) ordered trees, (iii) free trees, or (iv) hybrid trees (any combinations of (i), (ii) and (iii)). Alternative categorizations as shown in Table 2.4 could focus on the nature of the subtrees to be output (*induced subtrees*, *embedded subtrees*, *maximal subtrees*, or *closed subtrees*); or the nature of the support metrics employed (transaction-based counting denoted by T_c or occurrences-based counting denoted by O_c). In the table, the caption 'Maximal' ('Closed') denotes maximal (closed) frequent subtrees, the caption 'Induced' ('Embedded') denotes induced (embedded) subtrees. In the context of FTM, the definitions of maximal and closed frequent subtrees are similar to those of maximal and closed subgraphs (see Section 2.2), and induced subtrees and embedded subtrees are treated differently, which means {induced subtrees} $\not\subseteq$ {embedded subtrees} (see IMB3-Miner in Section 2.5.2). Further, in Table 2.4, the symbol '$\star$' is used to indicate the corresponding features for each FTM algorithm given in the first column. For an alternative review of FTM algorithms readers may like to refer to Chi et al. (2004), who provide a theoretical foundation and performance study of a representative collection of FTM algorithms proposed prior to 2004.

2.5.1 Rooted unordered tree mining

A labelled rooted unordered tree (see definition 2.1) is a labelled rooted tree where the left-to-right order among siblings is unimportant. It can be seen as a special form of *attributed relational graphs (ARG)*[8] and is effective in the modelling of structural data such as chemical compounds and the hyper-link structure of the Web [Asai et al.,

[7] IP (Internet Protocol) multi-cast is a method of building the multi-cast tree at the IP layer to send packets to multiple receivers in a single transmission [Paul, 1998]

[8] ARG is frequently used to describe structural objects where the nodes represent entities and the edges the relations between those entities [Tsai and Fu, 1979]

2003]. Seven rooted unordered tree mining algorithms are considered in this subsection: (i) TreeFinder, (ii) uFreqT, (iii) Unot, (iv) PathJoin, (v) cousinPair, (vi) RootedTreeMiner, (vii) SLEUTH.

(a) TreeFinder [Termier et al., 2002] employs the Apriori-based approach based on ancestor-descendant relationships to mine embedded subtrees. However, TreeFinder is an algorithm with inexact matching, which is only guaranteed to discover a subset of the complete set of frequent subtrees.

(b) uFreqT [Nijssen and Kok, 2003] uses a DLS representation to model unordered trees. At the candidate generate phase, the *rightmost path expansion* technique is employed to generate candidates. At the support counting phase, the tree mapping algorithm to determine the frequency of the pattern is translated into the maximum bipartite matching algorithm with complexity $\mathcal{O}(|E|\sqrt{|V|})$ (E and V are the edge and vertex sets of the bipartite graph). In order to facilitate the support computation, a data structure is used to store all potential mappings for the vertex on the rightmost path and pointers to the parent mapping.

(c) Asai et al. [2003] introduced an algorithm, Unot, that used a DLS to represent unordered trees. In Unot, the rightmost path expansion, supported by a *reverse search*[9] principle [Avis and Fukuda, 1996], was used to enumerate all the candidates without duplicates. However, Asai et al. [2003] only provided the theoretical basis for their algorithm.

(d) Xiao et al. [2003] introduced an algorithm, PathJoin, for mining maximal frequent induced subtrees in unordered tree databases. The algorithm used a data structure, *FST-Forest* which was inspired by Han et al. [2000], to compress the database. Based on FST-Forest, all maximal frequent paths were discovered, and then the frequent subtrees were generated by joining the frequent paths. After mining all frequent subtrees, the maximal frequent subtrees were produced by post processing. The reported evaluation of the algorithm was directed only at three synthetic data sets; no comparison was made with other algorithms.

(e) Shasha et al. [2004] presented an unordered tree mining algorithm, cousinPair, for application to phylogeny[10]. They defined an interesting pattern as being a "cousin pair", a pair of vertexes satisfying some cousin distance and minimum occurrence threshold. By using such constraints, interesting patterns were mined from a tree

[9] In reverse search, a parent-child relation is defined on the solution space of the mining such that each solution has a unique parent. Such relation can form an enumeration tree over the solution space. All the potential candidates can be generated starting from the root and computing the children for the root iteratively.

[10] Phylogeny: *"the evolutionary history of a taxonomic group of organisms"* [Biology-Online.org, 2010]

database. Shasha et al. noted that this kind of pattern can be used to get a better understanding of the evolutionary history of species. However, the algorithm was restricted to mining these special patterns only, and thus would seem not to have general applicability.

(f) Chi et al. [2005] presented an algorithm, RootedTreeMiner, to mine frequent induced subtrees. The BFCS encoding was employed to represent unordered trees. Thus, at the candidate generation phase, the range of allowable vertexes at a given position was computed beforehand in order to reduce the computational overhead required for the encoding. At the support counting phase, an *occurrence list* was built for each discovered subtree t. This list recorded the identifier (ID) of each transaction graph in the tree dataset that contained t, along with the mapping between the vertex indexes in t and those in the transaction. Using the occurrence list, the support of t can easily be calculated as the number of elements in the list with distinct IDs. In comparison with TreeMiner [Zaki, 2002], which will be discussed later in this Chapter, RootedTreeMiner was comparatively less efficient when the support was large, but performed well when the support was small.

(g) Zaki [2002] adopted the same techniques as used in previous subtree mining algorithms for ordered trees to propose the SLEUTH algorithm. This algorithm used a DFS-LS to represent unordered trees and scope-lists to compute the support. During the candidate generation phase, two extension mechanisms were employed: (i) class-based extension and (ii) canonical extension. For class-based extension, not all candidates generated by this mechanism were in the required canonical form, which meant that a process to ensure that each generated subtree was in the required canonical form was also needed prior to extension. For canonical extension the extension was only applied to the (canonical) frequent subtrees with known frequent edges, which resulted in many infrequent (but canonical) candidates. As indicated by Zaki [2005b], there was a trade-off between using the canonical extension and the class-based extension. Further reported experiments demonstrated that using class-based extension was more efficient than canonical extension.

2.5.2 Rooted ordered tree mining

In contrast to unordered tree mining, the inherent structure associated with ordered trees can be used to introduce efficiencies with respect to subtree generation and subtree isomorphism detection in ordered tree mining. Eight rooted ordered tree mining algorithms are reviewed in this sub-section: (i) FREQT, (ii) TreeMiner, (iii) Chopper, (iv) XSpanner, (v) AMIOT, (vi) IMB3-Miner, (vii) TRIPS, (viii) TIDES.

(a) In Asai et al. [2002], a collection of semi-structure data (e.g. Web pages) were modelled using a labelled ordered tree; and an algorithm, FREQT, introduced

to discover frequent subtrees from such data. The technique of *rightmost path expansion* was employed for the enumeration of candidates without duplicates, and only the rightmost leaf occurrences of the patterns were saved for efficient support counting. The reported evaluation of the algorithm indicated that it was useful for Web information extraction.

(b) Zaki [2002] adopted a DFS-LS representation to encode ordered trees, together with the concept of *equivalence class based extension*, to facilitate subtree generation. The notion of a *scope-list* was employed for fast support counting of discovered subtrees. Based on these concepts, the TreeMiner algorithm was introduced to discover a set of embedded subtrees. The performance of the algorithm was compared with a *base algorithm*, PatternMatcher, which employed a BFS strategy. The experimental results demonstrated that TreeMiner outperformed PatternMatcher using a real data set and that both algorithms scaled well when the number of trees in the data set was increased. However, the pruning technique adopted by TreeMiner is not as effective as PatternMatcher when using low support values. The reported results also indicated the usefulness of the discovered patterns with respect to Web usage mining.

(c) Wang et al. [2004a] proposed an algorithm, Chopper, to mine frequent embedded subtrees from a tree database. Firstly, the algorithm scanned the database to generate a sequential database comprised of a DLS representation for each tree. Secondly, a revised PrefixSpan algorithm [Pei et al., 2001b] was employed to mine frequent sequential patterns. Finally, the tree database was again scanned, against the discovered sequential patterns, to generate candidate patterns and find the frequent ones. Some additional overhead was needed for the algorithm, because the two processes of sequential pattern mining and subtree pattern verification were separated in Chopper. In order to improve the efficiency of Chopper, the XSpanner algorithm was subsequently produced to integrate the sequential pattern mining into the process of subtree pattern verification. Using projected database techniques, XSpanner grew larger frequent subtrees from smaller ones starting from one vertex. In comparison with TreeMiner, both Chopper and XSpanner outperformed TreeMiner when the support threshold was below 5%. However, XSpanner was found to be more stable than Chopper when using low support thresholds, and the former surpassed the latter when the support value was gradually decreased.

(d) Hido and Kawano [2005] noted that the enumeration using rightmost path expansion adopted by FREQT and TreeMiner still generated various non-frequent candidates which lead to unnecessary support counting, although it generated no redundant subtree candidates. Therefore, they introduced an algorithm, AMIOT, to utilize a new enumeration scheme to reduce the number of non-frequent can-

didates and keep the property of generating candidates only once. This scheme, *right-and-left tree join*, guaranteed that the set of subtree candidates was always a subset of that achieved by the enumeration using rightmost path expansion. The performance of AMIOT on synthetic data and an XML data set, demonstrated that it was scalable and performed faster than FREQT. However, the memory usage of AMIOT is larger than that of FREQT, due to the nature of the BFS strategy used by AMIOT.

(e) IMB3-Miner [Tan et al., 2006] was proposed to mine frequent embedded subtrees from an ordered tree database with a parameter to control the *level of embedding*[11]. When the level of embedding was equal to 1, the discovered frequent subtrees were induced subtrees. Thus, by adjusting the embedding level, the algorithm could mine both induced and embedded subtrees. By combining an *Embedding List* data structure with the *TMG* enumeration strategy (i.e. specialized rightmost path expansion), the algorithm guaranteed that candidate subtrees were generated efficiently without duplicates. Furthermore, an occurrence list was stored for each generated subtree to speed up the support counting. Instead of using T_c, O_c was employed to calculate the support of patterns. It has been experimentally demonstrated that IMB3-Miner achieves higher performance and scalability than TreeMiner and FREQT. Tan et al. [2006] also suggested that using O_c is necessary when the repetition and order of the patterns are important.

(f) Tatikonda et al. [2006] introduced a generic approach to mining induced or embedded subtrees in a database of rooted ordered trees. Their approach exploited two sequential encodings, prüfer sequencing (see 2.4.1) and DFS sequence, to represent the trees, and simplify the task of candidate subtree generation and frequency counting. Using prüfer sequencing and the leftmost path [12] of the pattern as the extension positions, the TRIPS algorithm was proposed, and alternatively the TIDES algorithm which used DFS sequence and rightmost path extension. The support computation for both algorithms employed an *embedding list*, an array based structure, to facilitate the recursive generation of the patterns. There is a trade-off between the cost of maintaining the embedding list and the efficiency of the support computation, when the number of distinct vertex labels was few compared with the total number of vertexes in the database. Experiments exhibited that both TRIPS and TIDES performed better than TreeMiner in terms of execution time and memory usage on both synthetic data and real data. Both TRIPS and TIDES were found to be scalable when the database size increased and could mine large databases at low support thresholds.

[11] The *level of embedding* is defined as "the length of path between two vertexes that form an ancestor-descendant relationship" [Tan et al., 2006].

[12] The path from the root to the left most leaf is the *left most path* [Tatikonda et al., 2006]

2.5.3 Free tree mining

As defined in Section 2.1, free trees are the connected, undirected and acyclic graphs. They are more expressive than paths and less expressive than general graphs. Thus, the mining of free trees is computationally faster than the mining of general graphs but harder than the mining of paths. Since free trees are less complex than general graphs, they have been utilized extensively, for example in domains such as bioinformatics [Hein et al., 1996], computer vision [Liu and Geiger, 1999], computer networking [Cui et al., 2005], and so on. Four such free tree mining algorithms are considered in this sub-section: (i) FreeTreeMiner, (ii) FTMiner, (iii) F3TM, (iv) CFFTree.

FreeTreeMiner [Chi et al., 2003a] was introduced to discover frequent subtrees in collections of free trees. A *self-join* operation was used for the candidate subtree generation and a subtree isomorphism algorithm [Chung, 1987] was implemented for the support computation. Experiments show that FreeTreeMiner can handle large real data well with a large range of support values. However, it was not found to be scalable when the size of the maximal frequent subtrees was increased due to the exponential growth of the number of potential frequent subtrees.

Ruckert and Kramer [2004] defined, independently, a canonical representation for labelled free trees, which was similar to that in [Chi et al., 2003a]. Accordingly, a free tree miner, FTMiner, was introduced to mine free tree patterns. The algorithm extended more than one vertex at each recursive step during the candidate generation phase. It also adopted the concept of an *extension table*, which was a data structure for storing all the extensions for a subtree pattern along with the set of transaction graphs containing the pattern. Utilizing this extension table, the algorithm not only kept track of the frequency of each subtree pattern, but also gathered information required for the extension of the current pattern, thus reducing significantly the number of database scans. Experiments on a large scale database suggest that the algorithm was able to mine frequent patterns within a collection of more than $37,330$ chemical compounds at a support threshold of 2%.

With the focus mainly on reducing the cost of candidate generation Zhao and Yu [2006] presented a free tree mining algorithm F3TM. The algorithm introduced the idea of an *extension frontier* to define the positions (vertexes) for growing frequent subtrees in the candidate generation phase, and used automorphism-based pruning and canonical pruning techniques to enhance the efficiency of candidate generation from the enumeration tree. Compared with other free tree mining algorithms, performance studies indicated that F3TM was more efficient than FTMiner and FreeTreeMiner using a chemical database of 42390 compounds. F3TM was further extended to introduce a closed frequent free tree miner, CFFTree [Zhao and Yu, 2007]. This algorithm employed *safe position pruning* to grow subtrees only from "safe" positions. In addition, this algorithm employed *safe label pruning* to grow subtrees only on the vertexes with labels

lexicographically less than the new vertex, which served to remove some unnecessary enumeration. The performance of CFFTree outperformed its base algorithm F3TM using post-processing to find the closed patterns.

2.5.4 Hybrid tree mining

Hybrid tree mining algorithms can be grouped according to the nature of the input tree data: (i) rooted ordered or unordered trees and (ii) rooted unordered or free trees. Each is briefly reviewed below.

2.5.4.1 Rooted ordered or unordered trees mining

CMTreeMiner was introduced to mine both *closed* and *maximal* frequent subtrees in collections of labelled rooted ordered or unordered trees [Chi et al., 2003b, 2004b]. The set of maximal frequent subtrees is a subset of the set of closed frequent subtrees, which in turn is a subset of the complete set of frequent subtrees. By using blanket[13] based pruning, in conjunction with a heuristic that determined the order of computing the blanket's subsets, the enumeration tree was grown only on the branches that were potentially able to produce closed or maximal frequent subtrees, thus avoiding the computational overhead associated with finding all frequent subtrees. Compared with the PathJoin algorithm, which mined maximal frequent subtrees by post-processing, the advantage offered by CMTreeMiner was that it directly mined closed and maximal frequent subtrees without first generating all frequent subtrees. Experimental results showed that: (i) for an ordered tree database, CMTreeMiner outperformed FREQT, (ii) for an unordered tree database, CMTreeMiner ran faster than HybridTreeMiner, and (iii) CMTreeMiner's memory usage was significantly smaller than PathJoin, and also the performance of the former was far better.

2.5.4.2 Rooted unordered or free trees mining

Chi et al. [2004c] presented the HybridTreeMiner algorithm to discover all frequent subtrees in a collection of labelled unordered trees or labelled free trees. This algorithm employed the same string encoding, BFCS, as used by Chi et al. [2005]. Inspired by the work of Huan et al. [2003], this algorithm combines the extension and join operations to efficiently generate subtree candidates. Within the enumeration tree that enumerates all subtree candidates based on their BFCFs, each node represented an unordered tree in BFCF. For a node v in the enumeration tree, all the children of v could be generated by either an extension or a join operation. The *join* operation was applied to a pair of sibling nodes with a height (depth) h, resulting in a BFCF tree with the same height; the *extension* operation was applied by extending a new leaf at the bottom level of the

[13]For a frequent subtree t, the *blanket* of t is defined as the set of frequent supertrees of t with one more vertex than t [Chi et al., 2003b]

BFCF tree with height h, resulting in a BFCF tree with height $(h+1)$. Additionally, the concept of an *equivalence relation* was introduced to efficiently enumerate all possible automorphisms[14] of a BFCF in the join operation.

This hybrid enumeration strategy was further extended to handle the free tree case. Reported experimental results demonstrated that HybridTreeMiner was faster than FreeTreeMiner, and that its memory usage was also much less than that required by FreeTreeMiner. As for unordered trees, although HybridTreeMiner runs faster than both Unot and uFreqT, it is hard to decide which algorithm is better because the margins are rather small.

Table 2.5: Main techniques used by frequent subtree mining algorithms

Algorithm	*Candidate Generation*	*Support Computation*
TreeFinder	Apriori itemset generation	clustering techniques
uFreqT	rightmost path expansion	maximum bipartite matching
SLEUTH	equivalence class extension	scope-lists
Unot	rightmost path expansion	embedding occurrence
PathJoin	FST-Forest	FST-Forest
cousinPair	cousin distance	lookup table
RootedTreeMiner	enumeration tree	occurrence list
FREQT	rightmost path expansion	occurrence list
TreeMiner	equivalence class extension	scope list join
Chopper XSpanner	n/a	n/a
AMIOT	right-and-left tree join	occurrence list
IMB3-Miner	TMG	occurrence list
TRIPS	leftmost path extension	hash table
TIDES	rightmost path extension	hash table
FreeTreeMiner	self-join	subtree isomorphism
FTMiner	extension tables	support sets
F3TM CFFTree	enumeration tree extension frontier	Ullmann's backtracking algorithm
CMTreeMiner	enumeration tree	n/a
HybridTreeMiner	extension + join	occurrence list

2.5.5 Summary of frequent subtree mining algorithms

From the foregoing it can be seen that for all the described FTM algorithms, many different methods, techniques and strategies have been proposed, as summarised in Table 2.5. This table characterizes the various techniques according to their adopted candidate generation and support counting mechanisms. It should also be noted that the reported FTM algorithms reviewed in this section were all evaluated using disparate data sets and compared with different alternatives, as shown in Table 2.6. There are various reasons for this including the diverse types of trees to be discovered and the various properties of the detected frequent subtrees, however this also means that

[14] An *automorphism* of a tree is the isomorphism of the tree to itself [Chi et al., 2004c]

it is difficult to make any direct and conclusive comparisons between the different techniques.

Table 2.6: A summary of frequent subtree mining algorithms

Algorithm	*Tree Representation*	*Dataset*	*Comparable Algorithm*
TreeFinder	relational formula	artificial XML data	n/a
uFreqT	DLS	n/a	n/a
SLEUTH	DFS-LS	Zaki's tree generator CSLOGS Dataset	TreeMiner
Unot	DLS	n/a	n/a
PathJoin	FST-Forest	Zaki's tree generator	n/a
cousinPair	n/a	synthetic dataset TreeBase	n/a
RootedTreeMiner	BFCS	synthetic dataset web access trees Zaki's tree generator	TreeMiner
FREQT	DLS	CiteSeer's web pages Allsite web pages	n/a
TreeMiner	DFS-LS	Zaki's tree generator CSLOGS	PatternMatcher
Chopper XSpanner	DLS	Synthetic data Web logs	TreeMiner
AMIOT	DFS string	Synthetic data XML data	FREQT
IMB3-Miner	DFS-LS	Synthetic data CSLOGS	FREQT TreeMiner PatternMatcher
TRIPS	CPS	Synthetic data CSLOGS TREEBANK dataset	TreeMiner XSpanner
TIDES	DFS sequence	Synthetic data CSLOGS TREEBANK dataset	TreeMiner XSpanner
FreeTreeMiner	BFCS	Synthetic data Chemical data Multicast data	n/a
FTMiner	n/a	anti-HIV data anti-Cancer data	AGM MolFea [Kramer et al., 2001]
F3TM	n/a	AIDS antiviral data	FreeTreeMiner FTMiner
CFFTree	n/a	AIDS antiviral data synthetic graphs	F3TM
CMTreeMiner	DFS-LS	Synthetic data CSLOGS Multicast data	FREQT HybridTreeMiner PathJoin
HybridTreeMiner	BFCS	Synthetic data web access trees chemical compounds	uFreqT FreeTreeMiner Unot

From the "view point" of the candidate generation methods employed in the mining process, all the FTM algorithms discussed in this thesis can be roughly categorized into three classes.

- Apriori-like generation
  In Apriori-like tree mining algorithms, candidate subtrees are generated level-wise by traversing a lattice structure of all frequent subtrees. The traversing strategy can be either in a DFS or BFS manner. At each level k, all candidate subtrees with size[15] k are generated from frequent subtrees discovered at level $k-1$ using a join or extension operation. Examples of algorithms using this generation method are TreeFinder, SLEUTH, TreeMiner, AMIOT, FreeTreeMiner, and HybridTreeMiner.

- Enumeration tree based generation
  In enumeration tree based tree mining algorithms, candidate subtrees with size $(k+1)$ are generated by extending their unique parent (frequent subtrees of size k in the enumeration tree). Examples of algorithms using this enumeration method are uFreqT, Unot, RootedTreeMiner, FreqT, IMB3-Miner, F3TM, CFFTree, CMTreeMiner, and HybridTreeMiner.

- FP-growth-like generation
  In this category, inspired by the FP-tree concept espoused by the FP-growth algorithm [Han et al., 2000] for ARM, a compact data structure is devised to facilitate mining frequent subtrees. An example of this type of algorithm is PathJoin.

From the "viewpoint" of the support counting methods employed in the mining process, all the FTM algorithms discussed in this thesis can be classified into the following three main groups.

- Occurrence list
  The occurrence list L for a k-subtree t records each occurrence of t in the dataset. Each member of L is of the form $[tid, v_1, v_2, \cdots, v_k]$, where tid is the transaction identifier in the dataset that contains t and $v_1, v_2, \cdots, v_k$ represent the mapping between the vertexes in t and those in the transaction tid. The support of t is the number of members in L with distinct tids. Examples of algorithms using occurrence lists are Unot, RooteTreeMiner, FREQT, IMB3-Miner, AMIOT, and HybridTreeMiner.

- Subtree isomorphism detection
  In this category, the support of a subtree t is computed directly when conducting subtree isomorphism detection. Examples of algorithms within this category are FreeTreeMiner, F3TM, and CFFTree.

- Special data structures
  This category covers algorithms that focus on special data structures to aid the

[15] The *size* of a tree is defined as the number of vertexes in the tree.

process of determining the support for each subtree. Some examples of such data structures include hash tables used by TRIPS and TIDES, scope lists used by TreeMiner and SLEUTH, FST-Forest used by PathJoin, and extension tables used by FTMiner.

Both the occurrence list and special data structures methods are devised to facilitate the expensive subtree isomorphism detection while the subtree isomorphism detection method directly utilizes existing subtree isomorphism detection algorithms to calculate the support. There are plenty of efficient subtree isomorphism detection algorithms that can be used in the literature.

From the perspective of the applications to which FTM is typically directed, the FTM algorithms discussed in this thesis can be divided into three main domains

- Web access trees
  Examples of algorithms are SLEUTH, RootedTreeMiner, TreeMiner, IMB3-Miner, Chopper, XSpanner, TRIPS, TIDES, CMTreeMiner, and HybridTreeMiner.
- IP multicast trees
  Examples of algorithms are FreeTreeMiner and CMTreeMiner
- Chemical compounds
  Examples of algorithms are FreeTreeMiner, FTMiner, F3TM, CFFTree, and HybridTreeMiner.

From the nature of the traversing strategy employed in the search space, all the FTM algorithms discussed in the thesis can be organized into two main categories

- BFS strategy
  BFS strategy based traversing has the advantage of performing full pruning, but at the cost of large memory usage. Examples of algorithms using this strategy are RootedTreeMiner, AMIOT, FreeTreeMiner, and HybridTreeMiner.
- DFS strategy
  DFS strategy based traversing has the disadvantage of weak pruning. However, the memory usage is smaller than that required using BFS strategies. Examples of algorithms using this strategy are uFreqT, Unot, SLEUTH, FREQT, TreeMiner, IMB3-Miner, TIDES, FTMiner, and CMTreeMiner.

As noted above, each tree mining algorithm has its strengths and weaknesses. There is no generic tree mining scheme, which can apply to any kinds of trees (unordered, ordered, and free), detect any types of subtrees (embedded, induced, closed, and maximal), and employ any frequency measures (T_c, and O_c). In terms of efficiency and effectiveness of the mining, the following four techniques are the most frequently quoted.

- DFS sequencing and its variants, frequently used for tree representation
- DFS strategy, usually used for traversing the search space
- Enumeration tree growth with rightmost path expansion, mostly used in candidate generation phase
- Occurrence lists, mainly used in support counting phase

Examples of algorithms containing at least three of these techniques are SLEUTH, FREQT, TreeMiner, and IMB3-Miner. Among these, FREQT, and TreeMiner are usually chosen as base algorithms for comparison with others. TreeMiner belongs to the Apriori-like group of FTM algorithms, while FREQT belongs to the rightmost path expansion group of algorithms. These two groups of algorithms represent two streams within the realm of FTM. Although subtree isomorphism can be solved in $\mathcal{O}(\frac{k^{1.5}}{\log k}n)$ time, surprisingly few FTM algorithms adopted it directly for support counting; occurrence lists are more frequently adopted. The main reason for this might be that occurrence list counting is much more straightforward to implement.

2.6 Frequent Subgraph Mining Algorithms

Frequent Subgraph Mining (FGM) algorithms have their origins in the identification of frequent patterns in chemical informatics and biological networks [Huan et al., 2004c, Deshpande et al., 2005, Fatta and Berthold, 2005, Hu et al., 2005, Wale and Karypis, 2006]. As with FTM algorithms, there are a wide variety of FGM algorithms reported in the literature. The identified two key phases associated with FTM algorithms (see 2.5), candidate generation and support counting, are also relevant with respect to FGM algorithms. For the candidate generation phase, the FGM algorithms seek to efficiently produce non-redundant candidates to the maximum extent possible. For the support counting phase, FGM algorithms seek to compute the frequency of the pattern without incurring the subgraph isomorphism detection excessively (in other words, FGM algorithms strive to avoid the subgraph isomorphism detection as much as possible). Since subgraph isomorphism detection is known to be NP-complete (see Section 2.3), a significant amount of research work has been directed at various approaches for achieving effective candidate generation, it is the nature of these differing approaches which best distinguish FGM algorithms from one another.

In this section, a review of a number of current well-known FGM algorithms is provided. Interested readers should note that a good review of the theoretical basis of FGM, prior to 2003, can be found in Washio and Motoda [2003]. A more recent overview of mining frequent patterns including itemsets, sub-sequences and sub-structures can be found in Han et al. [2007].

For the purpose of discussion, the FGM algorithms discussed in this thesis have been broadly categorized as follows:

(i) General purpose FGM.

(ii) Pattern dependent FGM.

The distinction is that in the latter case the nature of the patterns to be discovered is in some way specialized. Knowledge of the nature of these "special" patterns then allows for a reduction of the search space and the consequent computational effort required.

Table 2.7: General purpose frequent subgraph mining algorithm categorisation

Exact match based	transaction graphs	BFS strategy	AGM (Inokuchi 2000)
			AcGM (Inokuchi 2002)
			FSG (Kuramochi 2001)
			gFSG (Kuramochi 2002)
			DPMine (Gudes 2006)
		DFS strategy	MoFa (Borgelt 2002)
			gSpan (Yan 2002)
			FFSM (Huan 2003)
			GASTON (Nijssen 2004)
	one single graph	HSIGRAM (Kuramochi 2002)	
		VSIGRAM (Kuramochi 2002)	
		FPF (Schreiber 2005)	
		DPMine (Gudes 2006)	
Inexact match based	one single graph	SUBDUE (Cook 1994)	
		GREW (Kuramochi 2004)	
		gApprox (Chen 2007)	
		RAM (Zhang 2008)	

2.6.1 General purpose frequent subgraph mining

General purpose frequent subgraph mining algorithms may be further categorised (Table 2.7) into successive levels according to three criteria: (i) the completeness of the search, (ii) the type of input (transactions graphs or one single graph), and (iii) the search strategy. Table 2.7 presents an overview of a number of general purpose subgraph mining algorithms. For each algorithm the table gives only the first author's surname and year publication as the reference in order to save the space. Each algorithm will be discussed further in the following.

2.6.1.1 Inexact match based

Inexact match based algorithms use an approximate measure to compare the similarity of two graphs, i.e., two subgraphs are not required to be entirely identical to contribute to the support count, instead a subgraph in the input data may contribute to the support count for candidate frequent subgraph if it is in some sense similar to the

candidate subgraph. Algorithms using an inexact match approach are not guaranteed to find all frequent subgraphs, but the nature of the approximate graph comparison may lead to computational efficiency gains. There are few examples of inexact FGM algorithms in the literature. However, one well known and frequently quoted example is the SUBDUE algorithm [Cook and Holder, 1994, 2000]. SUBDUE uses the minimum description length principle [Grünwald, 2007] to compress the graph data; and a heuristic beam search method (starting with a single vertex), that makes use of background knowledge, to narrow down the search space. Although the application of SUBDUE shows some promising results in domains including image analysis and CAD circuit analysis, the scalability of the algorithm becomes an issue when dealing with complex data representations (i.e. the run time of SUBDUE is not linear with the size of the input graph). Furthermore, SUBDUE tends to discover a very small number of patterns.

Another inexact match based approach, GREW [Kuramochi and Karypis, 2004a] was directed at connected subgraphs which have many vertex-disjoint [16] embeddings in a single large graph. This heuristic based algorithm tended to be scalable to very large graphs because it employed the ideas of *edge contraction* and *graph rewriting* that underestimated the frequency of each discovered subgraph. Experiments on four benchmark datasets showed that GREW significantly outperformed SUBDUE with respect to runtime, the number of patterns found, and the size of the pattern.

To the best of the author's knowledge, the two most recent approaches regarding approximate patterns mining are gApprox, presented by Chen et al. [2007b], and RAM by Zhang and Yang [2008]. Chen et al. [2007b] used the notion of an upper-bound for support counting, and designed an approximation measure to discover frequent approximately connected subgraphs in a very large network. Empirical studies based on protein-protein interaction networks indicated that gApprox is efficient and that the discovered patterns were biological meaningful. Zhang and Yang [2008] presented a formal definition of frequent approximate patterns in the context of biological data represented by graphs, where the edge information was not accurate. Based on this, a randomized algorithm, RAM, using a hashing function was designed, the experiments showed that RAM can discover some important patterns which can not be found by exact match based mining algorithms. However, it is conjectured that RAM may be vulnerable to missing some important patterns.

2.6.1.2 Exact match based

Exact FGM algorithms are much more common. They can be applied in the context of transaction graph based mining or single graph based mining. A fundamental feature for exact match based algorithms is that the mining is complete, i.e. the mining

[16] Two embeddings in a graph G are *vertex-disjoint*, if they do not share any vertexes in G.

algorithms are guaranteed to find all frequent subgraphs in the input data. As noted in Kuramochi and Karypis [2004a], such complete mining algorithms perform efficiently only on sparse graphs with a large number of labels for vertexes and edges. To achieve completeness these algorithms must conduct a considerable amount of costly subgraph isomorphism checking (explicitly or implicitly).

2.6.1.2.1 Transaction graph based mining As noted previously, in transaction graph based mining the input is a collection of relatively small graphs. Algorithms within this class can be further divided into two classes: BFS strategy and DFS strategy, according to the strategy adopted for generating candidates. The BFS strategy is efficient for pruning infrequent subgraphs at the cost of high I/O and memory usage whereas DFS strategy needs less memory usage in exchange for less efficient pruning.

1. BFS strategy

   The BFS strategy tends to be used extensively as it extends the idea of the well-known "Apriori" algorithm for discovering frequent itemsets, which operates a level-wise search through the lattice of itemsets [Agrawal and Srikant, 1994]. The key element of Apriori lies in the fact that all subsets of a frequent itemset are frequent (i.e. DCP for itemsets). Using DCP infrequent patterns can be efficiently pruned. In analogy to Apriori, the BFS strategy based FGM algorithms search the lattice of subgraphs by utilizing the DCP for graphs, i.e. a $(k+1)$-subgraph can not be frequent if its immediate parent k-subgraph is not frequent.

   There are many examples of FGM algorithms that use the BFS strategy. One distinction between these algorithms is that the complete set of all k-candidates is discovered first before moving on to the set of $(k+1)$-candidates, where k refers to the expansion unit for growing the candidate subgraphs which can be expressed in terms of vertexes, edges, or disjoint paths. Four well-established algorithms, that fall within this category, are discussed below:

   (a) AGM

   The BFS strategy is exemplified by the AGM (Apriori-based graph mining) algorithm [Inokuchi et al., 2000] which is directed at finding frequent induced subgraphs. AGM uses an *adjacency matrix* to represent graphs and a level-wise search to discover frequent subgraphs. It assumes that all vertexes in a graph are distinct. The reported evaluation of AGM, on chemical carcinogenesis data, showed it to be more efficient than the inductive logic programming based approach combined with a level-wise search. AGM discovers not only connected subgraphs but also unconnected subgraphs with several isolated graph components. In consideration of real applications, a more efficient version of AGM called AcGM was proposed to mine only frequent connected graphs [Inokuchi

et al., 2002]. This algorithm used the same principles and graph representation as that of AGM, but with some improvements. Experimental results indicated that AcGM was significantly faster than AGM and FSG (see below). AGM was further extended Inokuchi et al. [2003] to mine frequent induced subgraphs from general graph datasets that can contain directed (or undirected), labelled (or unlabelled) graphs and even loops.

(b) FSG

FSG is another well established Apriori-like FGM algorithm proposed in Kuramochi and Karypis [2001, 2004b]. FSG is directed at finding all frequent connected subgraphs in a large graph dataset. FSG used a BFS strategy to grow candidates whereby pairs of identified frequent k-subgraphs are joined to generate $(k+1)$-subgraphs. FSG used a canonical labelling method for graph comparison and computed the support of patterns using *Transaction ID* (TID) lists (a vertical data representation). Each TID list represents a discovered subgraph along with the transaction identifiers (TIDs) of the graphs where it occurs (a very similar representation is adopted extensively in FTM algorithms). To determine the support for a k-subgraph, the intersection of the TID lists of its frequent $(k-1)$-subgraphs is computed first. If the size of the intersection is below the threshold, this k-subgraph is not frequent and can be pruned; otherwise the support is computed using subgraph isomorphism detection with respect to the intersection of relevant TID lists. TID list based support counting reduces the computation overhead. However, experiments show that FSG does not perform well when the transaction graphs contain many vertexes and edges that have identical labels because the *join* method used by FSG allows for multiple automorphism[17] of single or multiple cores[18]. In further experiments on chemical data, FSG required 600 seconds (with a 7% support threshold), while AGM needed 8 days (with a 10% support threshold) using the same database!

(c) gFSG

The above FSG algorithm is directed at graph datasets consisting of a two dimensional arrangement of vertexes and edges in each graph (sometimes referred to as topological graphs). However, in chemical compound analysis users are often interested in graphs having coordinates associated with the vertexes in two or three dimensional space (sometimes referred to as geometric graphs). Kuramochi and Karypis [2002] presented the gFSG algorithm to extend the FSG algorithm to discover frequent geometric subgraphs with some degree

[17] *Automorphism* is a graph isomorphism to itself via a non-identity mapping.

[18] In the candidate generation phase, a *core* is a common $(k-1)$-subgraph shared by two frequent k-subgraphs. Two frequent k-subgraphs are eligible for joining only if they contain the same core [Kuramochi and Karypis, 2001].

of tolerance among geometric transaction graphs. The extracted geometric subgraphs are rotation, scaling and translation invariant. gFSG shares the approach of candidate generation with FSG. In order to speed up the computation of geometric isomorphism, a number of topological properties and geometric transform invariants were used as keys for matching. In the process of support counting, geometric transform invariants (e.g. edge-angle lists[19]), and transaction lists are used to facilitate the computation. The reported experimental evaluation was performed on a chemical database with more than 20,000 chemical compounds (graphs) to show that gFSG operated well with low support values and scaled linearly with respect to the size of the database.

(d) DPMine

The AGM and FSG algorithms used vertexes and edges respectively as the expansion unit for candidate generation. In Vanetik et al. [2002] and Gudes et al. [2006], DPMine was proposed which used edge-disjoint paths as the expansion units for candidate generation. Use of such a large expansion unit reduces the number of candidates that are required for support counting. The algorithm firstly identified all frequent paths; secondly found all subgraphs with two paths; and thirdly merged pairs of frequent subgraphs with $(k-1)$ paths, which have $(k-2)$ paths in common, in order to obtain subgraphs with k paths. Experimental results indicated that support computation was the most significant contributor to the total computation time. Gudes et al. [2006] also suggested that reducing support computation overhead was more important than reducing the candidate generation overhead. It should be noted that the input for DPMine can be one single graph as well as a collection of transaction graphs.

2. DFS strategy

Algorithms in the DFS category need less memory, because they traverse the lattice of all possible frequent subgraphs in a DFS manner. Four example algorithms in this category are:

(a) MoFa

Proposed in Borgelt and Berthold [2002], MoFa was directed at mining frequent sub-structures (connected subgraphs) of molecules. The algorithm stored the embedding list of previously found subgraphs. The extension operation was restricted only to those embeddings which are actually in the database. MoFa also used structural pruning and background knowledge to reduce the support

[19] An edge-angle list is a multi-set where each element represents the angle formed by two distinct edges sharing the same end points.

computation. However MoFa still generated many duplicate candidate subgraphs, resulting in unnecessary support computation.

(b) gSpan

gSpan [Yan and Han, 2002] used a canonical representation, M-DFSC, to uniquely represent each subgraph. The algorithm used DFS lexicographic ordering to construct a tree-like lattice over all possible patterns, resulting in a hierarchical search space called a DFS code tree. Each node of this search tree represented a DFS code. The $(k+1)$-th level of the tree had nodes which contain DFS codes for k-subgraphs. The k-subgraphs were generated by one edge extension from the k-th level of the tree. This search tree was traversed in a DFS manner and all subgraphs with non-minimal DFS codes were pruned so that redundant candidates are avoided. Instead of keeping embedding lists for each discovered subgraph, gSpan only preserved the transaction list for each discovered pattern and subgraph isomorphism detection was only applied to graphs within the list. Thus, gSpan saved on memory usage because keeping embedding lists required a lot of memory resources to store all embeddings of a subgraph in the database. Reported experiments demonstrated that gSpan outperformed FSG by an order of magnitude. gSpan is one of the most frequently quoted FGM algorithms. Because of its popularity, gSpan was adopted as the "base" algorithm into which many of the proposed weighted FSM techniques described in this thesis were incorporated for evaluation purposes.

(c) FFSM

Many FGM algorithms, such as FSG and gSpan, can efficiently handle graphs which are relatively small and have a large number of different vertex labels. However they operate less well on graphs that are large and dense with a small number of labels as found in, for example, protein structure mining. Huan et al. [2003] proposed a new algorithm, FFSM (Fast Frequent Subgraph Mining), to focus on such graphs. FFSM adopted the CAM representation (see above). Thus, a tree-like structure, a suboptimal CAM tree, was constructed to include all possible patterns. Each node in this suboptimal CAM tree can be enumerated by either a join or an extension operation. FFSM recorded embedding lists for each discovered pattern to avoid explicit subgraph isomorphism testing during the support counting phase. Performance evaluation undertaken by Huan et al. [2003], using several chemical data sets, indicated that FFSM outperformed gSpan. FFSM was adopted as one of the algorithms used to evaluate the proposed weighted FSM techniques described in this thesis.

(d) GASTON

Nijssen and Kok [2004] observed that most frequent sub-structures in molecular databases are free trees. Based on this observation, they introduced GASTON,

which integrated frequent path tree and graph mining into one algorithm. The algorithm divided the FGM process into three phases: path mining, then subtree mining, and finally subgraph mining. Consequently the subgraph mining was only invoked when needed. Thus, GASTON operated best when the graphs are mainly paths or trees, because the more expensive subgraph isomorphism testing is only encountered in the subgraph mining phase. GASTON used a hash function and graph isomorphism detection to identify the duplicates. GASTON also recorded the embedding lists so as to only grow patterns that actually occurred in the transaction graphs, thus saving on unnecessary isomorphism testing. The reported experiments indicated that the operation of GASTON compared favourably with a wide range of functionally comparable miners, GASTON was able to scale up to the real datasets comprised of some 250,000 transaction graphs. GASTON was also used later in the research reported in this thesis to compare results produced using the proposed weighted FSM schemes.

Because of the diversity of the different transaction graph based FGM algorithms, it is difficult to present a clear picture of the strong and weak points associated with the various algorithms. However, Wörlein et al. [2005] presented a detailed comparison of four DFS-based miners: MoFa, gSpan, FFSM, and GASTON, with respect to their performance on various chemical databases. In the reported experiments, they found that using embedding lists did not remarkably accelerate the search for frequent patterns. They also confirmed that using canonical representations for duplicate detection required less computation than explicit subgraph isomorphism detection.

2.6.1.2.2 Single graph based mining Recall that for single graph based mining, the input to the mining algorithm is a large single graph. The frequency of a pattern is determined by the number of its occurrences in the entire graph, compared to transaction graph based mining where an individual transaction graph contributes only once to a frequency count even though there may be multiple occurrences in the same transaction. Single graph based mining can be applied to transaction graph based mining, but the latter can not be applied to the former. An fundamental issue regarding single graph based mining is how to define the support of the pattern. The *DCP*, often used to prune the search space when using transaction-based support, does not hold in the case of occurrence-based support. Alternative occurrence-based support measures that satisfy the *DCP* are therefore desirable. One seminal class of occurrence-based support measure that maintains the *DCP* is based on the idea of an *overlap graph*[20].

[20] An *overlap graph* for a given pattern with a set of all embeddings (occurrences) is a constructed graph where each vertex represents a non-identical embedding of the pattern and two vertexes are connected if the corresponding embeddings overlap [Kuramochi and Karypis, 2005].

By building a overlap graph for each pattern using the number of its occurrences in the entire graph, the occurrence-based support measure is defined as the size of the maximum independent set (MIS) of vertexes in the overlap graph. The MIS measure was first introduced in Vanetik [2002] (see also Kuramochi and Karypis [2004c, 2005]). In Vanetik et al. [2006], the formal definition was provided together with the proofs for the sufficient and necessary conditions required for occurrence based support measures to maintain the *DCP*. Their work was further extended to introduce a new occurrence based support measure, which kept the *DCP*, and was computable in polynomial time [Calders et al., 2008]. Four main examples of single graph based FGM algorithms are introduced in the following:

(a) HSIGRAM, VSIGRAM, and FPF

Kuramochi and Karypis [2004c, 2005] proposed a number of solutions for finding all frequent subgraphs in a large sparse graph. Two algorithms, HSIGRAM and VSIGRAM, were introduced to mine all subgraphs whose embeddings were edge-disjoint in one large sparse graph. These two algorithms were based on the BFS and DFS strategy respectively. The support of each discovered subgraph was determined by the overlap based MIS measure, which maintains the *DCP* [Vanetik et al., 2006]. Several variations of the MIS measures, including exact MIS and approximate MIS measures, were implemented. Experiments demonstrated that both algorithms scaled well when mining large graphs, although VSIGRAM ran faster than HSIGRAM. The reason for the performance advantage of the VSIGRAM algorithm is that it kept track of the embeddings of the frequent subgraphs along the DFS path, resulting in less subgraph isomorphism computation. In comparison with SUBDUE, the results indicated that SUBDUE performs worse than both HSIGRAM and VSIGRAM, and that SUBDUE tends to focus on small subgraphs with high frequency, thus missing significant patterns. Kuramochi and Karypis [2004c]'s work was further extended by Schreiber and Schwöbbermeyer [2005] to mine frequent patterns of a given size, but considering different frequency concepts. This different frequency based algorithm, FPF, was applied to two different biological networks to discover network motifs[21]. Surprisingly, the comparison of the values (i.e., number of frequent patterns) for the different frequency concepts showed that the frequency of a pattern alone was not sufficient to identify network motifs. Furthermore, whether frequent patterns can play functional roles in the biological network is still unclear, according to Schreiber and Schwöbbermeyer [2005].

(b) DPMine

DPMine [Vanetik et al., 2002, Gudes et al., 2006] has already been described under

[21] *Network Motifs* are defined as "patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks" [Milo et al., 2002].

the heading of transaction graph based mining (see Sub-section 2.6.1.2). The algorithm can also be applied to single graph based mining and is sometimes described as a single graph based mining algorithm.

2.6.2 Pattern dependent frequent subgraph mining

The foregoing discussions assumed that the user wished to find all frequently occurring subgraphs that exist in either a single large graphs or a collection of medium sized transaction graphs. This is of course not necessarily the case, users may be interested in a certain type of pattern, i.e. some subset of all the frequent subgraphs. Such "special patterns" can be defined according to their topology and/or some specific constraints on the nature of the patterns. Five groups of special patterns can be identified from the literature. These are itemized in Table 2.8. Each of the associated algorithms will be discussed in more detail in this section.

Table 2.8: Categorisation of pattern dependent frequent subgraph mining algorithms

Relational Patterns	CLOSECUT [Yan et al., 2005c] SPLAT [Yan et al., 2005c]
Maximal Patterns	SPIN [Huan et al., 2004b] MARGIN [Thomas et al., 2006]
Closed Patterns	CloseGraph [Yan and Han, 2003]
Clique Patterns	CLAN [Wang et al., 2006] Cocain [Zeng et al., 2006]
Constrained Patterns	gPrune [Zhu et al., 2007]

2.6.2.1 Mining relational patterns

Relational graphs are suitable for modelling large scale networks such as biological or social networks. Yan et al. [2005c] noted that the mining of relational patterns has three features that serve to differentiate it from general purpose FGM: (i) distinct vertex labels, (ii) very large size graphs, and (iii) a focus on frequent patterns with certain *connectivity constraints* (e.g. the minimum degree of a pattern[22]). Thus relational graph mining aims to identify all frequent patterns that display some specific connectivity constraint.

Two example algorithms, CLOSECUT and SPLAT, both proposed by Yan et al. [2005c], are directed at mining closed frequent subgraphs with connectivity constraints in a database of relational graphs. The CLOSECUT algorithm used a pattern-growth approach to integrating connectivity constraints, together with graph condensation and decomposition techniques. The SPLAT algorithm used a pattern reduction approach to integrating the graph decomposition technique. Experiments indicated that CLOSECUT performed better than SPLAT on patterns with high support and low

[22]The *minimum degree* of a pattern g is the minimum of the degree of v, for all $v \in V(g)$ [Yan et al., 2005c].

connectivity; SPLAT performed better than CLOSECUT on patterns with high connectivity and low support. The results using biological databases showed that both algorithms could find interesting patterns with strong biological meanings.

2.6.2.2 Mining maximal patterns and closed patterns

As noted previously the number of possible frequent subgraphs increased exponentially with the size of the graph, i.e. for a frequent k-graph, the number of frequent subgraphs can be as large as 2^k. An example given in Yan and Han [2003] indicated that about $1,000,000$ frequent graphic patterns could be generated from 422 chemical compounds, using a support threshold of 5%; and among these many were found to be structurally repetitive. Therefore, both *closed* and *maximal* FGM approaches have been proposed as mechanisms to reduce the number of frequent subgraphs generated. These approaches are discussed below. The following notation is used: MFS denotes the set of maximal frequent subgraphs, CFS the set of closed frequent subgraphs, and FS the set of all frequent subgraphs in the graph database. Thus, $MFS \subseteq CFS \subseteq FS$.

2.6.2.2.1 Mining maximal patterns Let the set of maximal frequent subgraphs $MFS = \{g | g \in FS \wedge \nexists h \in FS$ such that $g \subset h\}$. Thus a maximal frequent subgraph is a graphic pattern such that all of its subgraphs are supported but none of its supergraphs are supported. The task of maximal FGM is to find all graphic patterns that belong to the MFS. Maximal frequent subgraphs encode the maximal common structures, which are deemed to be the most interesting patterns in (for example) the domain of biological networks [Koyutürk and Szpankowski, 2004]. However, the frequency of non-maximal subgraphs is not produced, which requires scanning of the original database. Two example algorithms where the mining of maximal patterns technique has been incorporated are:

(a) SPIN

The SPIN algorithm, presented by Huan et al. [2004b], is a spanning tree based FGM algorithm designed to discover only maximal frequent subgraphs with the intention of reducing the overall computational cost. In SPIN, the concept of canonical spanning tree[23] based equivalence classes was first applied; and then, a graph partitioning method, founded on tree-based equivalence classes integrated with three pruning techniques, was used to mine maximal subgraphs. The algorithm has two main phases: (i) identification of all frequent subtrees within the input data using an appropriate FTM algorithms; (ii) detection of all frequent subgraphs whose canonical spanning tree was isomorphic to each discovered frequent subtree. The maximal frequent subgraphs were then computed. The performance

[23] A *canonical spanning tree* of a graph is defined as the lexicographically maximal spanning tree of the graph [Huan et al., 2004b].

of SPIN was compared with gSpan and FFSM. The results showed that SPIN gave a significantly better performance than gSpan and FFSM using both synthetic and chemical data. In addition, SPIN had good scalability with respect to large graph datasets.

(b) MARGIN

Thomas et al. [2006] proposed the MARGIN algorithm to mine maximal frequent subgraphs. The algorithm was based on the observation that the set of potential maximally frequent subgraphs is included in the set of frequent k-subgraphs that have infrequent $(k+1)$-supergraphs. Consequently the search space of MARGIN was significantly reduced by pruning the lattice around the set of potential maximally frequent subgraphs. The set of subgraph candidates is recursively discovered by the core algorithm, *ExpandCut*, and the maximal frequent subgraphs were then found by the post-processing operation. Experimental results showed that MARGIN was computationally faster than gSpan when applied to some data sets. However, the efficiency of MARGIN largely relied on the initial *cut*[24] to be chosen.

2.6.2.2.2 Mining closed patterns Let the set of closed frequent subgraphs $CFS = \{g | g \in FS \wedge \nexists h \in FS$ such that $g \subset h \wedge support(g) = support(h)\}$. Thus, given a frequent subgraph g, if there exists no supergraph h such that every transaction graph containing g also contains h, then g is a closed frequent subgraph. The task of closed FGM is to find all such graphic patterns. In the context of biochemistry and chemical compound analysis these closed patterns are of interest because biochemists are often interested in the largest fragment with certain properties [Fischer and Meinl, 2004].

Yan and Han [2003] noted that in the domain of chemical compound classification not all frequent subgraphs in the graph database are effective patterns with respect to classification accuracy. They showed that using $2,000$ closed frequent subgraphs achieved the same accuracy as when all $1,000,000$ frequent subgraphs were used. Inspired by this fact, they adopted the idea of mining closed frequent itemsets or sequences (e.g. [Zaki and Hsiao, 2002, Wang and Han, 2004]) to devise a closed FGM algorithm, called CloseGraph. This algorithm was founded on gSpan and used *equivalent occurrence based early termination* to prune the search space. For the case where early termination failed and could not be applied, detection of the failure of early termination was implemented. The experimental results showed that CloseGraph was found to perform better than gSpan and FSG. Yan and Han also emphasized that a better failure detection was needed in order to improve CloseGraph, because the algorithm's performance is mainly dependent on the efficiency of the failure detection.

[24] A *cut* between two nodes in the lattice is defined as an ordered pair (p, c) where node p represents the parent of node c and p is infrequent while c is frequent [Thomas et al., 2006].

2.6.2.3 Mining clique patterns

A clique[25] (or quasi-clique) is a subset of one subgraph with a fixed topology. Since the first algorithm introduced by Harary and Ross [1957] to detect cliques, a significant number of algorithms have been devised for a variety of the clique detection problems [Bomze et al., 1999, Gutin, 2004]. Recently researchers have established that discovering frequent cliques from a set of transaction graphs is useful in domains such as communication, finance, and bioinformatics. Example applications where the mining of cliques, or quasi-cliques, has been applied include: community mining [Abello et al., 2002], gene expression mining [Pei et al., 2005], and the discovery of highly correlated stocks from stock market graphs [Wang et al., 2006]. General purpose FGM algorithms can be used to discover such "special patterns", however the computation can be made more efficient if the special properties of cliques is taken into account. Two examples of clique mining algorithms: (i) CLAN and (ii) Cocain are discussed in the following paragraphs.

(a) CLAN

CLAN is an algorithm proposed to mine frequent closed cliques from large dense graph databases [Wang et al., 2006]. The algorithm utilized the properties of the clique structure to facilitate clique or sub-clique isomorphism testing by introducing a canonical representation of a clique. Wang et al. [2006] devised several pruning techniques to effectively reduce the search space. For efficiency evaluation purposes CLAN was compared with ADI-Mine [Wang et al., 2004b] using US stock market and chemical compounds data sets. The results showed that CLAN ran efficiently with respect to large and dense graph databases. However, the reported efficiency experiments only used high support thresholds, and the scalability was only demonstrated using a small and sparse graph database.

(b) Cocain

Extending the work of CLAN, a general form of clique mining algorithm, Cocain, was introduced to mine closed γ-quasi-cliques[26] from large and dense graph databases [Zeng et al., 2006]. In Cocain, Zeng et al. [2006] restricted the quasi-cliques by satisfying a user specified parameter, γ, which has to be over 0.5. They further utilized the properties of quasi-cliques to devise several search space pruning techniques, combined with a closure checking scheme, to speed up the discovery process. However, the empirical evaluation was limited to only US stock market databases.

[25] Let $V(g)$ denote the set of vertexes in a graph g, a subset $s \subseteq V(g)$ is a *clique* if the subgraph induced on s is a complete graph.

[26] A γ-*quasi-clique*$(0 \leq \gamma \leq 1)$ is a k-subgraph $(k \geq 1)$, g, where $\forall v \in V(g)$, $degree(v) \geq \lceil \gamma(k-1) \rceil$ [Zeng et al., 2006]

2.6.2.4 Mining constrained patterns

In real applications, it is often desirable for the user to control the characteristics of the patterns to be discovered. General purpose FGM, as discussed above, can not push user-defined constraints into the mining process. A straightforward solution is to firstly mine all frequent patterns using an appropriate general purpose FGM algorithm, and then examine each discovered frequent pattern in the context of the desired constraints. However, the nature of the constraints on the solution can be usefully employed to enhance the efficiency of the discovery process.

User constraint based frequent pattern mining has been thoroughly studied in the domain of ARM (e.g. [Ng et al., 1998]). The main idea is to integrate constraints into the mining process in order to prune the search space. This idea was further extended by Bonchi et al. [2003] to consider the search spaces for both patterns and data, using the pruning of both anti-monotone and monotone constraints. Inspired by this, Zhu et al. [2007] presented a framework, called gPrune, to incorporate various constraints into the FGM process. The reported empirical study showed that the effectiveness of integrating constraints into the mining process is influenced by many aspects, including the properties of the data and the pruning cost. The design of such constraint based mining algorithms needs to take into consideration the trade-off between the pruning cost and the potential benefit.

2.6.3 Summary

Tables 2.9 and Table 2.10 present a summary of the FGM algorithms discussed in this Section. Table 2.9 itemizes, for each algorithm: the representation used, the nature of the data sets to which the algorithm has been applied and the reported algorithms with which each has been compared. Table 2.10 gives the candidate generation and support counting mechanism associated with each. As can be seen from the tables, as in the case of FTM, the nature of FGM is extremely diverse. Consequently it has not been possible to present a complete overview of all FGM algorithms in this thesis. However, in the literature, SUBDUE, AGM, FSG, MoFa, gSpan, FFSM, and GASTON are the most frequently cited. Among these algorithms, SUBDUE is used more widely than others. One major disadvantage of SUBDUE is that the algorithm tends to find small size patterns and miss important patterns. AGM and FSG are two representative BFS-based miners. MoFa is a specialized miner for molecular databases and is able to mine directed graphs. FFSM and GASTON can not be used for directed graphs while gSpan can accommodate directed graphs if some minor changes are incorporated.

One common feature associated with the majority of algorithms described above is that the search space for the mining is usually modelled as a tree-like lattice over all possible patterns, which are ordered lexicographically. Each node in the lattice represents one pattern, and the relationship between patterns at $(k+1)$-level and k-level only

Table 2.9: A summary of the FGM algorithms reported in this thesis

Algorithm	*Graph Representation*	*Input Data*	*Comparable Algorithms*
AGM/AcGM	CAM	synthetic data chemical data	n/a
FSG	CAM	synthetic data chemical data	n/a
gFSG	n/a	chemical data	n/a
DPMine	n/a	synthetic data XML data email traffic data	FSG
MoFa	n/a	chemical data	n/a
gSpan	M-DFSC	synthetic data chemical data	FSG
FFSM	CAM	synthetic data chemical data	gSpan
GASTON	n/a	synthetic data chemical data	gSpan FSG FreeTreeMiner FTMiner
HSIGRAM VSIGRAM	CAM	aviation data credit data contact map data chemical data VLSI data	SUBDUE
FPF	CAM	biological data	n/a
DPMine	n/a	synthetic data XML data email traffic data	FSG
CLOSECUT	M-DFSC	synthetic data biological data	SPLAT
SPLAT	n/a	synthetic data biological data	CLOSECUT
SPIN	BFCS	synthetic data chemical data	gSpan FFSM
MARGIN	n/a	synthetic data chemical data	gSpan
CloseGraph	M-DFSC	synthetic data chemical data	gSpan FSG
CLAN	vertex label sequence	US stock market	n/a
Cocain	vertex label sequence	US stock market	n/a
gPrune	M-DFSC	synthetic data biological data	n/a

differs by one vertex or edge, resulting in a 'parent-child' relation. The search is then translated into traversing the lattice and keeping all patterns satisfying the threshold. In the lattice, a BFS or DFS strategy can be used to traverse the lattice. For BFS strategy based miners, the candidate subgraphs are constantly generated from smaller ones that have a common *core*. More specifically, $(k+1)$-subgraphs are generated by joining two frequent k-subgraphs which have a common core of $(k-1)$-subgraph. This level-wise join operation is the same as for most BFS strategy based miners (also

Table 2.10: Main techniques used by FGM algorithms reported in this thesis

Algorithm	*Candidate Generation*	*Support Computation*
AGM/AcGM	level-wise join	database scan
FSG	level-wise join	transaction list
gFSG	level-wise join	edge-angle list transaction list hybrid
DPMine	level-wise join	n/a
MoFa	extension	embedding list
gSpan	rightmost path extension	transaction list
FFSM	join + extension	embedding list
GASTON	path,tree, and graph enumeration	embedding list
HSIGRAM	level-wise join	various MIS measures
VSIGRAM	extension	various MIS measures
FPF	extension	MIS measure
DPMine	level-wise join	n/a
CLOSECUT	rightmost path extension	transaction list
SPLAT	n/a	n/a
SPIN	join	embedding set
MARGIN	ExpandCut	n/a
CloseGraph	rightmost path extension	transaction list
CLAN	DFS-based extension	n/a
Cocain	DFS-based extension	n/a
gPrune	rightmost path extension	transaction list

called Apriori-based algorithms); however the expansion unit tends to vary (e.g. vertexes, edges, or edge-disjoint paths). In comparison with DFS strategy based miners, BFS strategy based miners can obtain a tight upper bound for the support of the k-subgraphs using the support associated with the complete set of identified $(k-1)$-subgraphs. This upper bound can be employed to prune infrequent candidates. DFS strategy based miners typically derive an upper bound for k-subgraphs based only on a single $(k-1)$ parent frequent subgraph, which leads to less pruning.

As indicated in Wörlein et al. [2005], an efficient frequent subgraph miner usually contains three distinct features:

- Restrictive extension: The extension of a subgraph is valid only when the extension exists in the graphs within the subgraph's occurrence list. Examples of such operations are rightmost path extension by gSpan and extension by MoFa.

- Efficient candidate generation: This operation is achieved by applying a canonical graph representation so as to facilitate the filtering out of duplicates candidates before performing graph isomorphism. Two main canonical representations are CAM used by AGM, FSG, FFSM; and DFS code used by gSpan.

- Essential subgraph isomorphism: When computing the support of a pattern, a trade-off needs to be sought between explicitly using subgraph isomorphism and

storing embeddings of the pattern. Examples of algorithms that store embeddings are FFSM and GASTON, and instances of using subgraph isomorphism include FSG and gSpan.

Although FGM algorithms have many applications in various domains, there are still some issues that need to be addressed:

- **Scalability** The inherently exponential complexity of FGM means that low support values result in the discovery of large sets of patterns and correspondingly high run-times. Wang et al. [2004b] proposed an index for the scalable mining of large disk-based graph databases. Fatta and Berthold [2005] presented a distributed approach to the FGM problem. They extended MoFa to accommodate the distributed computation of mining frequent patterns from very large molecular compounds. Although both approaches can handle the scalability issue, they circumvent this issue by using multiple computer processors or an efficient index; they do not address the inherent exponential issue itself. GREW is the first algorithm presented to reduce the inherently exponential complexity. However, the input to the algorithm is one single undirected graph, not a collection of graphs.

- **Other generic interestingness measures** Current FGM algorithms restrict the notion of interestingness only to the support metric. Many other interestingness measures, other than the support measure, have been proposed (e.g. interestingness metrics for association patterns [Tan et al., 2002]). Also, in many cases, a single support metric may be too general to define interestingness with respect to individual applications, often too many similar patterns are generated. Xin et al. [2006c] proposed finding interesting frequent patterns in tabular data according to the user's "real" interest. However, this work is based on the assumption that the set of frequent patterns have already been mined. In Huan et al. [2004c], a coherent subgraph miner was proposed to identify interesting subgraphs using the *mutual information* based measure. Although the patterns discovered by this coherent subgraph miner have been proved to be discriminative in classifying protein structures, Huan et al. [2004c] did not show that these patterns can be applied to other domains. In addition, the above solutions are mainly applicable to the chemical domain and need some additional information provided by the user. There is little research work on adopting generic interestingness measures to assess the importance of the subgraph patterns.

- **Analysis of the resultant set of frequent subgraphs** Using relatively low support values, the resultant set of identified frequent subgraphs is very large, which means that further analysis of the patterns is extremely challenging. In Chen et al. [2008], a solution that approximates a complete set of frequent subgraphs using a small compact pattern set, is proposed to overcome this issue.

Patterns contained in this small compact set have the properties of high significance and low redundancy. However, this solution is considered to belong to the post-processing category and relies on the structural properties of the patterns and the quality measure. Thus, an advanced solution that can directly discover such a small compact pattern set is desirable.

Since there are an increasing number of interaction graphs which evolve over time, such as co-authorship networks, there is a growing interest in mining dynamic graphs [Desikn and Srivastava, 2004, Leskovec et al., 2005, Berger-Wolf and Saia, 2006]. The discovery of frequent subgraphs among dynamic graphs has been little explored. One example is given in Borgwardt et al. [2006], who propose extending the FGM algorithm to discover frequent patterns among dynamic graphs. The experimental results obtained by applying this approach to an email network represented by time series graphs validates the feasibility of the approach. However, the size of each time-stamp graph is relatively small.

2.7 Classification Using Frequent Subgraphs

One common application of using frequent patterns discovered by the algorithms introduced in Sections 2.5 and 2.6 is to classify a set of graphs. The procedure of how to classify graphs is explained in the next paragraph. Additionally, as described in Section 1.4, the patterns discovered by the proposed weighted FSM algorithms in the thesis are evaluated in a framework of frequent pattern based classification, to assess the quality of the discovered patterns. Therefore, it is necessary to give a brief description of the research work related to classifying of graphs.

The application of kernel methods [Schölkopf and Smola, 2002] for graph structured data (i.e. graph kernels [Gärtner et al., 2003, Kashima et al., 2003, Borgwardt and Kriegel, 2005]) has been widely used for the task of classifying a set of graph represented molecules in the domain of chemical informatics. However, such task can also be performed using a frequent pattern based classification framework. The general idea is illustrated in Figure 2.9 (a). In the figure, 'FSM' denotes the frequent subgraph mining process, 'SI' denotes subgraph isomorphism detection, and 'FS' denotes feature selection. Figure 2.9 (b) illustrates the process of constructing feature vectors. As indicated in Figure 2.9 (b), each graph is represented as a feature vector $G_i = \{f_1, f_2, \cdots, f_n\}$ where f_i is the frequency of the i-th subgraph pattern which occurs in that graph. The f_i is usually computed by conducting subgraph isomorphism detection between the i-th subgraph and each G_i in the database. In a binary scenario, f_i can be the existence or absence (1 or 0) of the i-th subgraph pattern. Each graph is associated with a class label at the end of the vector. Because the construction of feature vectors require subgraph isomorphism detection, assigning binary values for f_i is considerably faster than

assigning the actual frequency for f_i. In addition, for certain graph represented data (e.g. images), using the actual frequency of the pattern is more important than using the existence of the pattern in the image classification task. When building feature vectors for data sets described in this thesis, the actual frequency is used for the image data and the binary values are used for the rest of data.

After constructing these vectors for graphs, various classification approaches, including support vector machines, decision trees, and naive Bayesian classifiers can be employed to categorize these graphs. Examples of such features based classification algorithms include: frequent subgraph based classifier ([Deshpande et al., 2005]), coherent[27] frequent subgraph based classifier ([Huan et al., 2004c]), and closed frequent subgraph based classifier([Liu et al., 2005]). All these algorithms used frequent subgraphs as features for classification purpose, however they differ in the techniques adopted for feature selection as described in Figure 2.9(a). Because use of all identified frequent subgraphs as features is not feasible, all these algorithms extract highly discriminative features from the complete set of frequent subgraphs so that high quality classification is achieve without incurring a high computational overhead. One major disadvantage of these approaches is the separation of the FGM and feature selection process.

Several approaches have been proposed recently to address this drawback [Kudo et al., 2004, Nowozin et al., 2007, Yan et al., 2008, Thoma et al., 2009, Jin et al., 2009]. One common feature of these approaches is that they integrate the process of FGM and the selection of the discriminative features. Kudo et al. [2004] and Nowozin et al. [2007] proposed using the boosting method to discover frequent patterns in multiple iterations. For each iteration, the misclassified graph was weighted by a higher value. However, high classification accuracy is often achieved at the cost of a long execution time. Yan et al. [2008] proposed a binary classifier that used the top-k frequent patterns whose significances were evaluated by an objective function. However, as indicated by Jin et al. [2009], the performance of this classifier may be compromised when there are few discriminative patterns and the set of top-k patterns share most of their supporting graphs. Furthermore, the use of the objective function depends on the known class labels, which are not easy to obtain in real applications. CORK [Thoma et al., 2009] is another frequent subgraph based binary graph classifier, whose goal is to discover the most discriminative frequent subgraph set by using some quality measure. This CORK measure has a sub-modular feature which enables CORK to achieve good results. However this measure may fail to measure the discriminative strength of the set of frequent subgraphs under certain circumstance as noted in Jin et al. [2009]. COM, proposed by Jin et al. [2009], is the latest approach to using frequent subgraph

[27] A graph is *coherent* if the mutual information between it and each of its subgraph is greater than or equal to some threshold.

patterns for binary graph classification. By using FFSM as the base algorithm, COM groups subgraph patterns discovered by FFSM into co-occurrence rules by applying a simple quality measure during the mining. Thus, such rules can be used for classifying graphs. Experiments show that COM is more efficient and effective than the previous approaches proposed by Yan et al. [2008] and Thoma et al. [2009].

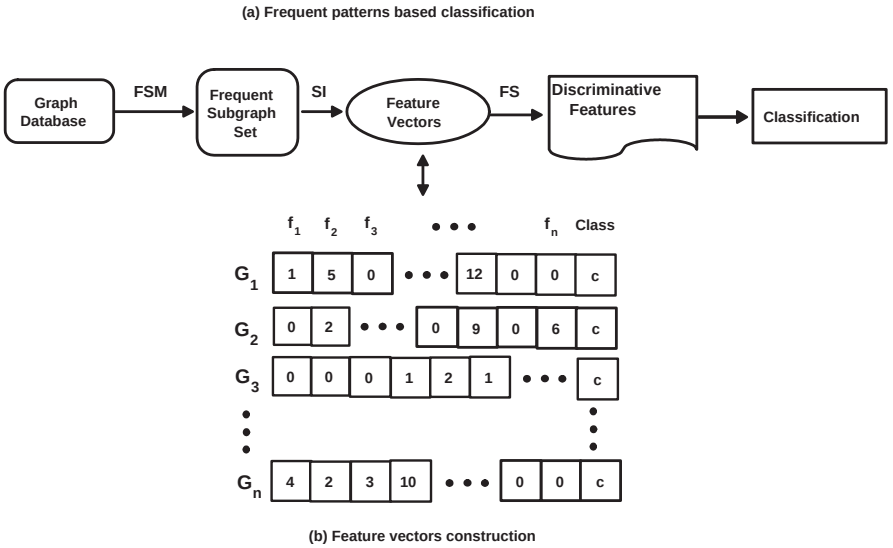


Figure 2.9: Frequent pattern based classification

Since different subgraphs can be discovered by various frequent subgraph mining algorithms, using such subgraphs as features may obtain different clustering and classification results. Thus, as indicated by [Han and Kamber, 2006], clustering and classification of sets of graphs should be coupled tightly with the mining of frequent subgraphs. The significance of frequent pattern based classification with respect to this thesis is that it is the mechanism adopted to evaluate the proposed weighted FSM schemes.

2.8 Social Network Analysis

A social network is naturally represented as a large graph or a set of transaction graphs where the transaction graph corresponds to a time-segmented network. Thus, the social network analysis task can be simply transformed into a task of mining interesting patterns in the graph data. How to mine the patterns in graph represented social networks has recently attracted some research interest.

Social networks comprise a set of social entities (nodes) and the relations between them (links). Social network analysis (SNA) ([Wasserman and Faust, 1994]) aims to examine the relations among the social entities, social structures, social positions and other related tasks. Because of the increased growth of the Internet, many large scale social networks have been studied such as scientific co-authorship networks ([Newman, 2003]), email communication networks ([Diesner et al., 2005]), and mobile call networks

([Nanavati et al., 2006]). As indicated by [Chakrabarti and Faloutsos, 2006], most of these large scale networks share three common features: (i) scale free distribution [Faloutsos et al., 1999, Redner, 1998]; (ii) small world effect [Watts and Strogatz, 1998, Albert et al., 1999]; (iii) strong community structure [Flake et al., 2000, Girvan and Newman, 2002]. Social network analysis has attracted a lot of research interest in a variety of tasks including centrality analysis ([White and Smyth, 2003]), community detection ([Newman and Girvan, 2004]), link prediction ([Nowell and Kleinberg, 2007]) and many others. Among these, community detection plays a fundamental role in SNA.

Generally, community detection can be defined as finding groups of nodes in a social network, where such nodes are linked more frequently within a group than between groups. Various definitions and approaches are exploited for community detection in different context ([Girvan and Newman, 2002, Newman, 2004a,b, Flake et al., 2004, Wu and Huberman, 2004]).

All the above approaches to SNA concentrate on mining patterns in the network represented by one large graph. Only a few researchers have shown interest in using frequent subgraph mining to identify patterns (i.e. frequent subgraphs) in the network represented by a set of transaction graphs, due to issues associated with efficiency and scalability. One example is the work of Lahiri and Berger-Wolf [2007], where frequent subgraphs were used to predict the partial structure in the temporal network at an unseen time-step. However, in Lahiri and Berger-Wolf [2007], due to the constraints imposed on the temporal network, the task of mining frequent subgraphs is simplified to the task of mining frequent itemsets, which greatly alleviates the computation cost. One category of graph data used for the evaluation in this thesis, which will be described in Sub-section 3.2.8, represents a social network as a set of time-stamped graphs. For evaluation purposes, the weighted frequent subgraph mining algorithms proposed in this thesis have been applied to these graphs to discover weighted frequent subgraphs.

2.9 Weighted Frequent Subgraph Mining

Since the research work proposed in this thesis (see Section 1.2) aims to use weighted FSM to reduce the computation complexity incurred by FSM, the research work that is closely related to the proposed weighted FSM is discussed in this section. Firstly, a variety of solutions proposed by other researchers that can lessen the combinatorial complexity of FSM are examined and their deficiencies are further analysed. Secondly, some further research work that has an influence on the proposed weighted FSM is discussed. In order to highlight the link between the existing research work and the research work proposed in this thesis, and the motivation of the proposed research work, a significant part of content discussed in Section 1.1 is recapitulated in the following paragraphs.

Research work on pattern discovery has advanced from mining itemsets and se-

quences to trees and graphs. In the context of frequent subgraph mining, various approaches have been proposed and numerous strategies haven studied as described in Sections 2.5 and 2.6. The "bottleneck" for frequent subgraph mining algorithms is the computational complexity incurred by the two core operations: candidate generation and support counting. Three further issues are: (i) interpretation the resulting set of patterns is often difficult; (ii) choosing the most appropriate minimum support threshold is not easy, and (iii) the generation of (unnecessary) structurally repetitive patterns. Four potential solution have been introduced to address these issues.

(1) Mining maximal or closed frequent subgraphs [Yan and Han, 2003, Huan et al., 2004b, Thomas et al., 2006]

(2) Mining approximate frequent subgraphs [Kelley et al., 2003, Sharan et al., 2005, Chen et al., 2007b]

(3) Summarizing the result set of patterns [Xin et al., 2006a, Chen et al., 2008]

(4) Mining frequent subgraphs with constraints [Zhu et al., 2007]

Among these four research directions, the first three have been examined substantially. Most of the approaches employed can find their counterparts in the domain of ARM (many techniques, strategies and methods used in FSM can be traced back to ARM).

In constraint based FSM, there is a little work on using a weight constraint. The research described in [Zhu et al., 2007] is directed at mining frequent subgraphs that satisfy some global graph properties such as size, density and diameter[28]. The most related work on weight constraint based FSM can be found in Eichinger et al. [2008]. Eichinger et al. [2008] used weights to adjust the importance of discovered patterns in the task of software bug detection. They considered the process of weight constraint based FSM as a two-stage procedure: (i) mining frequent subgraphs and (ii) post-processing the set of frequent subgraphs with weights. Consequently the success of their approach relied on the performance of FSM. This thesis takes the view that weightings should be integrated into the mining process so that efficiencies can be introduced. To the best knowledge of the author, little research work has been reported with regard to utilizing weightings within the process of FSM.

For studies directed at social networks (as described in Section 2.8), the connections in the network are usually assumed to be binary valued (either present or absent). However, the social network data (see Sub-section 3.2.8) used for evaluation purposes in this thesis, are assumed to have pre-defined edge weightings. Therefore, it is worthwhile to report on current research directed at the mining of weighted networks. As indicated by Wasserman and Faust [1994], it is common for real-world networks such as email

[28]The *diameter* of a connected graph is the maximum distance between any two vertexes in that graph [Chartrand and Zhang, 2004].

networks [Ebel et al., 2002], social networks [Kossinets and Watts, 2006], and scientific collaboration networks [Barrat et al., 2004], to be weighted inherently by assigning different strengths, intensities or capacities to their vertexes or edges. For instances, in a social network there may be stronger or weaker social relations between individuals; in a transportation network, the weight of the connection may be calculated by the amount of traffic flowing along it [Barrat et al., 2004]. There are a few studies [Newman, 2001, Yook et al., 2001, Barrat et al., 2004, McGlohon et al., 2008] directed at investigating the statistical properties of edge-weighted networks.

As mentioned earlier in Section 2.8, finding strongly connected groups of nodes in the network (i.e. community detection) is a common task in SNA [Snijders, 2001, Watts and Strogatz, 1998]. Nevertheless, such community detection has been mostly pursued by researchers without considering the significance (i.e. weights) of particularly nodes and/or edges [Ding et al., 2001, Shi and Malik, 2000, Newman, 2004b, Newman and Girvan, 2004, Clauset et al., 2004]. Newman [2004c] provides a solution to mining community structures in weighted networks by adapting existing community detection algorithms. The rationale is that for a weighted network, a weighted graph representation can be mapped onto an un-weighted multi-graph so that most existing community detection algorithms on the un-weighted graph can be applied to the weighted case with minor modification.

Although little work has been reported in the field of weighted FSM, there is a substantial body of reported work directed at weighted ARM (WARM) and weighted sequence mining (WSM) [Cai et al., 1998, Wang et al., 2000, Tao et al., 2003, Yun and Leggett, 2005, 2006, Yun, 2007]. A significant issue in WARM or WSM is that the DCP of itemsets or sequences, on which most ARM or sequence mining algorithms are based, no longer holds. Cai et al. [1998] firstly introduced the idea of a weighted support bound to address WARM, which is in spirit similar to utility mining. One solution adopted by Wang et al. [2000], is to handle the weights as a post-processing step after mining frequent itemsets, however the weights are not integrated into the ARM process. Tao et al. [2003] proposed another solution, using a framework of weighted support, which satisfied the DCP, to mine the most significant patterns and consequently reduce the overall computational complexity. Yun and Leggett [2005, 2006] and Yun [2007] introduced a series of concepts such as"weight range", "weight confidence" and "support confidence" (very similarly to *h-confidence* as proposed by Xiong et al. [2006] in correlated itemsets mining) in the mining of weighted frequent itemsets and sequences, in order to maintain the DCP. Within their framework, the correlated patterns (i.e. itemsets or sequences) with similar weight and support levels, termed *weighted affinity patterns*, are discovered by integrating weight confidence and support confidence into the base itemset or sequence mining algorithm.

Although the ideas espoused by WARM can not be directly applied to weighted

FSM, the research work described in the thesis is at least partially influenced by this body of work.

Algorithm 2.3: gSpan-Miner(c, σ, $\mathbb{GD}$, F)

Input: c = a subgraph represented by a DFS code, σ = minimum support, $\mathbb{GD}$ = a graph dataset

Output: $\mathcal{F}$, a set of frequent subgraphs

```
1  sort labels of the vertexes and edges in GD by their frequency
2  remove infrequent vertexes and edges
3  relabel the remaining vertexes and edges in descending frequency
4  F_1 ← {all frequent 1-edge subgraphs in GD}
5  sort F_1 in DFS lexicographic order
6  F ← ∅
7  foreach c ∈ F_1 do
8  |   subgSpan(c, GD, σ, F)
9  |   GD ← GD − c
10 |   if |GD| < σ then
11 |   |   break
12 |   end
13 end
```

2.10 Summary

The essential knowledge underpinning the research work proposed in this thesis is presented in this chapter. Firstly, the definitions used throughout the thesis were introduced in Section 2.1, and the proposed research problem was defined in Section 2.2. Secondly, the primary operations involved in FSM were discussed in Sections 2.3 and 2.4, and the principal approaches to FSM were analysed in Sections 2.5 and 2.6. Thirdly, the frequent pattern based classification framework was explained in Section 2.7. This framework was used to evaluate the quality of the patterns discovered by the proposed weighted FSM algorithms. Fourthly, since social network data sets were also used to test the proposed research work described in this thesis, the research work related to using FSM to discover patterns in social networks was introduced in Section 2.8. Finally, other research that has an influence on the work described in this thesis was discussed in Section 2.9, where it was noted that there has been very little previous work directed at the weighted FSM problem.

Three standard FSM algorithms that discussed in Section 2.6.1.2: gSpan, FFSM, and GASTON were used for evaluating the proposed weighted FSM schemes in this thesis. Among these three, gSpan was chosen as the base algorithm so that subgraph weighting schemes could be integrated into the algorithm to devise weighted FSM algorithms; FFSM and GASTON algorithms were used as baseline algorithms for comparing with the proposed weighted FSM algorithms. A detailed description of each algorithm

by pseudo-codes is presented in the following sub-sections.

2.10.1 Pseudo-codes of gSpan

The pseudo-codes of the gSpan algorithm following the description in Yan and Han [2002] are provided in Algorithm 2.3, which recursively call a "subgSpan" procedure.

Procedure subgSpan(c, $\mathbb{GD}$, σ, $\mathcal{F}$)

```
1  if c ≠ min(c) then
2  |  return
3  end
4  F ← F ∪ {c}
5  C ← ∅
6  Scan GD once, find every edge e such that c can be right-most extended to c ∪ e,
   C ← c ∪ e
7  Sort C in DFS lexicographic order
8  foreach g_k ∈ C do
9  |  if support(g_k) ≥ σ then
10 |  |  subgSpan(g_k, GD, σ, F)
11 |  end
12 end
```

2.10.2 Pseudo-codes of FFSM

The pseudo-codes of the FFSM algorithm following the description in Huan et al. [2003] are demonstrated in Algorithm 2.4, which recursively call a "FFSM-Search" procedure.

Algorithm 2.4: FFSM-Miner(C, σ, $\mathbb{GD}$, $\mathcal{F}$)

Input: C = a suboptimal CAM list, σ = minimum support, $\mathbb{GD}$ = a graph dataset

Output: $\mathcal{F}$, a set of frequent subgraphs

```
1  F ← {the CAMs of the frequent vertexes and edges}
2  F_1 ← {the CAMs of the frequent edges}
3  FFSM-Search(F_1, F)
```

2.10.3 Pseudo-codes of GASTON

An overview of the GASTON algorithm which is following the description in Nijssen and Kok [2005] is given in Algorithm 2.5. In Algorithm 2.5, the $refinement$ operation (line 1) is achieved by extending the current structure by the addition of both a new vertex and an new edge to connect this vertex (referred to as a node refinement) or the addition of a new edge which only connects existing vertexes (referred to as a cycle closing refinement); a *leg* of a structure in Nijssen and Kok [2005] is defined as a

Procedure FFSM-Search(W, $\mathcal{F}$)

Input: W = a suboptimal CAM list

Output: $\mathcal{F}$, a set of frequent subgraphs

```
1  foreach P ∈ W do
2      if P.isCAM then
3          F ← F ∪ {P}
4          C ← ∅
5          foreach Q ∈ W do
6              C ← C ∪ FFSM-Join(P, Q)
7          end
8          C ← C ∪ FFSM-Extension(P)
9          remove CAM(s) from C that is either infrequent or not suboptimal
10         FFSM-Search(C, F)
11     end
12 end
```

frequent refinement of a structure which also records the embedding list. In line (2) of Algorithm 2.5, the current graph g' is determined to see whether or not it is a canonical graph. If it is not, then g' is not further refined.

Algorithm 2.5: Gaston-Miner(g, l, L, σ, $\mathbb{GD}$, $\mathcal{F}$)

Input: g = a graph, a leg l, a set of legs L, σ = minimum support, $\mathbb{GD}$ = a graph dataset, $\mathcal{F}$ = a set of frequent subgraphs

Output: $\mathcal{F}$, a set of frequent subgraphs

```
1  g' ← refinement(g, l)
2  if g' is not canonical then
3      return
4  end
5  F ← F ∪ g'
6  L' ← {l' | l' is a necessary leg of g' ∧ support(refinement(g', l'), GD) ≥ σ ∧ (l' ∈
   L ∨ l' connects to the node introduced by l, if l is a node refinement)}
7  for ∀l' ∈ L' do
8      Gaston-Miner(g', l', L', σ, GD, F)
9  end
```

Since the GASTON algorithm divides the mining into three phases: paths, trees and graphs, Algorithm 2.6 presents a description of the GASTON algorithm by three different phases [Nijssen and Kok, 2004]. Further details can be found in Nijssen and Kok [2004, 2005].

Algorithm 2.6: Outline of the three phases of the Gaston-Miner: paths, trees, and cyclic graphs

```
1  Gaston-Miner_path(a path P, a set of legs L)
2  foreach allowable refinement leg l ∈ L do
3  |  g' ← refinement(P, l)
4  |  if l.refinement is a node refinement then
5  |  |  L' ← extend(l) ∪ {join(l, l')|l' ≠ l ∈ L}
6  |  |  if g' is a path then
7  |  |  |  Gaston-Miner_path(g', L')
8  |  |  else
9  |  |  |  Gaston-Miner_tree(g', L')
10 |  |  end
11 |  else
12 |  |  L' ← L' ∪ {join(l, l')|l' ≠ l ∈ L}
13 |  |  Gaston-Miner_cyclicgraph(g', L')
14 |  end
15 end
16 Gaston-Miner_tree(a tree T, a set of legs L)
17 foreach allowable refinement leg l ∈ L do
18 |  g' ← refinement(T, l)
19 |  if l.refinement is a node refinement then
20 |  |  L' ← restricted-extend(l) ∪ {join(l, l')|l' ∈ L ∧ l' is allowable in g'}
21 |  |  Gaston-Miner_tree(g', L')
22 |  else
23 |  |  L' ← L' ∪ {join(l, l')|l' ≠ l ∈ L}
24 |  |  Gaston-Miner_cyclicgraph(g', L')
25 |  end
26 end
27 Gaston-Miner_cyclicgraph(a graph G, a set of legs L)
28 foreach allowable refinement leg l ∈ L do
29 |  g' ← refinement(G, l)
30 |  L' ← L' ∪ {join(l, l')|l' > l ∈ L}
31 |  Gaston-Miner_cyclicgraph(g', L')
32 end
```

Chapter 3

Graph Data Sets

Various data sets were used for the evaluation of the weighted frequent subgraph mining algorithms proposed in this thesis. These data sets may be divided into two categories: (i) synthetic data sets, and (ii) real application data sets. The majority of the data sets included class labels so that classification algorithms could eventually be applied and used to evaluate the proposed weighting techniques. Among the synthetic data sets, some were selected because they are widely used by researchers in the domain of frequent subgraph mining; others were specifically constructed, by the author, for the purpose of conducting the desired evaluation. For the real application data sets, each selected data set represented a potential target domain of application for individual proposed frequent subgraph mining algorithms. A variety of mechanisms were adopted, according to the nature of the data, to allow the data sets to be represented as graphs. Some of these mechanisms had been proposed by other researchers, the remainder were developed by the author. The rest of this chapter is organized as follows. The synthetic data sets are introduced in Section 3.1 and the real application data sets in Section 3.2. Section 3.3 then summarizes all the data sets employed in this thesis.

3.1 Synthetic Data Sets

In this section, five different synthetic data sets, all represented in the form of trees, are introduced. The first two were generated by other researchers, and the remaining three by the author's research group members. According to the techniques used to model the data sets in the form of trees, the data sets were divided into two groups, ST1 and ST2:

(1) **Synthetic Tree 1 (ST1)**: Synthetic trees created by a random tree generator; two tree data sets.

(2) **Synthetic Tree 2 (ST2)**: Synthetic images represented by quad-trees; three tree data sets.

Each group will be discussed in further detail in the following sub-sections respectively.

3.1.1 ST1 - Synthetic trees created by a random tree generator

In the ST1 group of data sets the tree data was created, using the random tree generator, proposed in Zaki [2002], to simulate website browsing behaviour. The tree generator first generates a master tree, $\mathcal{T}$, controlled by the first four parameters shown in Table 3.1, and then creates a number of subtrees of $\mathcal{T}$ as specified by the parameter t. Further details of this tree generation process can be found in Zaki [2005a].

Table 3.1: Synthetic tree data set parameters

Notation	*Description*
f	The maximum degree of a node
d	The maximum depth of the tree
m	The total number of nodes in the master tree
n	The number of node labels
t	The number of subtrees created

Two synthetic data sets were generated. The first used the same parameter values as that described in Zaki [2002]: $n = 100$, $m = 10000$, $d = 10$, $f = 10$, $t = 100000$. The second replaced the value of t with $t = 1000000$. The characteristics of these data sets is given in Table 3.2.

Table 3.2: Characteristics of the synthetic "web usage" data sets

	D10	T1M
# trees (t)	100000	1000000
Max # edges	94	94
Average # edges	3	3
Max # vertexes	95	95
Average # vertexes	4	4
# Vertex labels	100000	1000000
# Edge labels	1	1

3.1.2 ST2 - Synthetic images represented by quad-trees

The ST2 group of data sets comprised synthetic trees, expressed using a quad-tree representation [Finkel and Bentley, 1974], to model a sequence of synthetic images generated by a random image generator [Coenen, 2009]. Quad-trees are a widely used tree structure for the representation of images. The basic idea of this representation, which is illustrated in Figure 3.1, is that any image can be divided into quadrants. Each quadrant can then be further split into equal-sized sub-quadrants, and so on until some imposed limit is met.

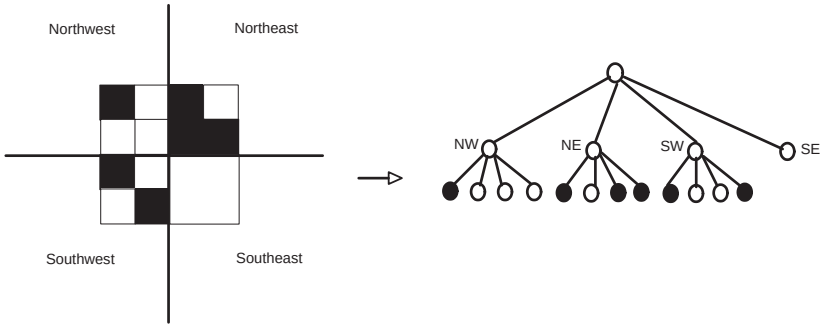
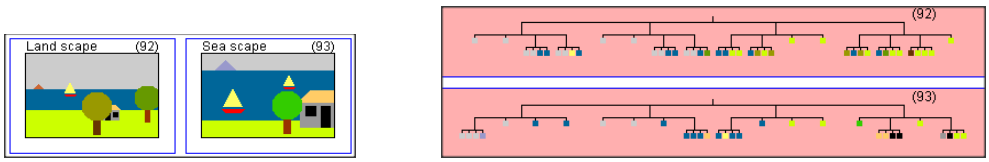


Figure 3.1: An illustration of the quad-tree representation. The image on the left is recursively divided into sub-quadrants (NW, NE, SW and SE).

Examples of the images generated by the random image generator and the subsequent quad-tree representations are shown in Figure 3.2. In Figure 3.2(a) two images, generated by the random image generator, are shown. The associated quad trees are shown in n Figure 3.2(b). For the purpose of evaluating the work described in this thesis, three quad-tree represented image data sets were generated using a sequence of quad-tree levels[1] of 4, 5, and 6. The characteristics of these three data sets are presented in Table 3.3. In the table, 'IM1000' indicates a set of 1000 images, and the symbol 'D' indicates the level of the quad-tree. These 1000 images are evenly divided into two classes (Seascape vs. Landscape).



(a) Random images (b) Quad tree representation

Figure 3.2: An illustration of quad-tree represented random images

Table 3.3: Characteristics of quad-tree represented image data sets

	IM1000-D4	IM1000-D5	IM1000-D6
# trees	1,000	1,000	1,000
Max # edges	76	208	524
Average # edges	34	87	192
Max # vertexes	77	209	525
Average # vertexes	35	88	193
# Vertex labels	18	18	18
# Edge labels	4	4	4

[1]The maximum number of leaf nodes in any quad-tree is decided by 4^L, where L is the user specified decomposition threshold used during the generation process

3.2 Real Datasets

In this section, a number of real graph and tree represented data sets, drawn from genuine applications and used for evaluation purposes in this thesis, are described. These real application data sets are categorized into eight groups according to the nature of the application in which they are likely to be used:

(1) **Real Application Tree 1 (RT1)**: Web usage mining.

(2) **Real application Tree 2 (RT2)**: Magnetic Resonance Imaging (MRI) brain scan analysis.

(3) **Real Application Tree 3 (RT3)**: Text mining founded on document semantics.

(4) **Real Application Graph 1 (RG1)**: Chemical compound analysis.

(5) **Real Application Graph 2 (RG2)**: Mammography.

(6) **Real Application Graph 3 (RG3)**: Photo classification.

(7) **Real Application Graph 4 (RG4)**: Text mining founded on term occurrence.

(8) **Real Application Graph 5 (RG5)**: Social network mining.

The RT1 and RG1 data have been widely used within the frequent subgraph mining community, for benchmark testing; and therefore were selected for inclusion in this thesis. The rest of six groups were acquired specifically for the purpose of testing the performance of the proposed algorithms. Each category is discussed in further detail in the following sub-sections.

3.2.1 RT1 - Web usage mining scenario

The RT1 group of data sets consisted of the CSLOGS data sets used in the studies described in [Zaki, 2002, 2005a, Zaki and Aggarwal, 2006]. These data sets comprise web usage log files with respect to the Computer Science department website hosted at Rensselaer Polytechnic Institute[2]. The structure and content of each "user session" is described, using the Log Markup Language (LOGML) [Punin et al., 2001], in terms of a *user browsing tree*. The CSLOGS data comprises three individual data sets: (i) CSLOGS-ALL, (ii) CSLOGS-1, (iii) CSLOGS-2. CSLOGS-ALL [Zaki, 2002, 2005a] consists of more than one month of web log files; while CSLOGS-1 and CSLOGS-2 describe a two week sequence of log files with each data set holding one week's worth of data. CSLOGS-ALL contains 59691 user browsing trees, but no class labels. However, the browsing trees in the smaller CSLOGS-1 and CSLOGS-2 data sets [Zaki

[2] `http://www.cs.rpi.edu/`

and Aggarwal, 2006] are divided into into two class: *edu* (denoting users from an "edu" domain) and *other* (denoting users from any other domain). The characteristics of the RT1 (CSLOGS) group of data sets are presented in Table 3.4.

Table 3.4: Properties of tree based CSLOGS web usage data sets

	CSLOGS-ALL	CSLOGS-1	CSLOGS-2
# trees	59691	8074	7409
Max # edges	429	314	172
Average # edges	13	9	9
Max # vertexes	430	315	173
Average # vertexes	14	10	10
# Vertex labels	59691	15652	14413
# Edge labels	1	1	1
# Class 1	n/a	1962	1686
# Class 2	n/a	6112	5721

3.2.2 RT2 - MRI brain scan images represented by quad-trees

The next three groups of real application data sets focused on image mining. The data sets were drawn from three different image mining domains: (i) Magnetic Resonance Imaging (MRI) brain scan analysis, (ii) Mammographic images, and (iii) Photographic images. Two different approaches to modelling these images, in terms of graphs, were considered:

- **Quad-tree representation.** As described in Sub-section 3.1.2.
- **Image interest points based representation.** In a region based image representation, the identified regions and relations between regions are encoded using an Attributed Relational Graph (ARG) format [Tsai and Fu, 1983, Schalkoff, 1992], where the vertexes represent regions, and the edges the relation between pair of regions. Similar to the region based image representation where image regions are captured using image segmentations, the image interest points based representation is an alternative form of the region based image representation where image regions are captured using salient interest points[3] [Tuytelaars and Mikolajczyk, 2008]. The claimed advantage of using image interest points is that it bypasses many of the errors incurred when image segmentation process are adopted.

The MRI images were represented using quad-trees and the remaining two using image interest points. The MRI scan data set is considered further in this sub-section,

[3]Interest points are important image features that contain high information of an image [Yang et al., 2007]

while the remaining two real application data sets are considered in the following two sub-sections respectively.

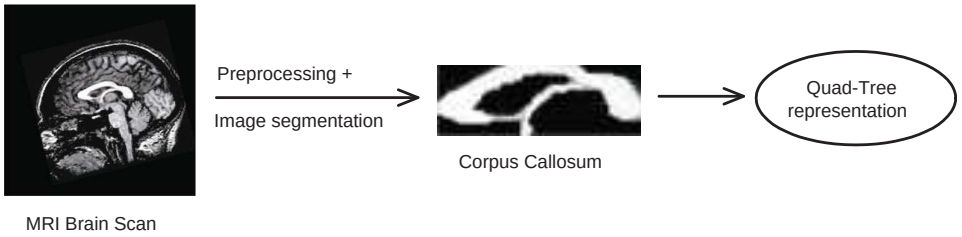


Figure 3.3: The procedure for generating quad-tree represented MRI brain scan images

Magnetic Resonance Image (MRI) brain scanning is a medical imaging technique, used in radiology, to produce detailed pictures of the brain and surrounding nerve tissues [MedlinePlus, 2010]. The RT2 group of data sets, described in Elsayed et al. [2010], contain 106 MRI brain scan images, which are equally divided into two classes: (i) musicians and (ii) non-musicians. The data set was originally collected to support the conjecture that the corpus callosum (a distinctive feature in MRI brain scans) is different for musicians than for non-musicians [Elsayed et al., 2010]. The processing of the image set is presented in Figure 3.3. After necessary image preprocessing, the corpus callosum in each image was fitted into a Minimum Bounding Rectangle (MBR). The given MBR space was then recursively decomposed into quadrants. Each quadrant was represented by a vertex in the quad-tree (with colour black or white), with the root of the quad-tree representing the entire MBR space. The decomposition process was terminated when a predefined level of granularity was reached, or a particular sub-quadrant was sufficiently homogeneous (95% black or white). The three data sets contained in the RT2 group of data sets were all generated in this manner, but using three different quad-tree decomposition level ranging from 5 to 7 (equivalent to a maximum number of vertexes of 1024, 4096, and 16384 respectively). Note that the degree of detail increases with the level of the quad-tree decomposition. The characteristics of RT2 data sets employed in this thesis are given in Table 3.5.

Table 3.5: Properties of RT2 data at different quad-tree levels

	QT-D5	QT-D6	QT-D7
Quad-tree Level	5	6	7
# Trees	106	106	106
Max # edges	196	208	476
Average # edges	117	149	282
Max # vertexes	197	209	477
Average # vertexes	118	150	283
# Vertex labels	3	3	3
# Edge labels	4	4	4

3.2.3 RT3 - Text mining founded on document semantics

The sixth group of data sets were directed at text mining applications. The data sets were extracted from: Medline[4], the International Movie Database (IMDB)[5], Amazon[6], and Oshumed[7]. The data sets were represented as graphs using two approaches: (i) semantic graph based and (ii) term occurrences based, as indicated in Table 3.6. In the table, the symbol '$\surd$' in each cell indicates which text representation approach is adopted for which text data collection. The RT3 group of data sets comprised the semantic graph based group of data sets and is discussed further in this sub-section. The RG4 group of data sets comprised the term occurrence based group of data sets and is discussed in Section 3.2.7.

Table 3.6: Categorisation of the text data sets according to the graph representation method adopted

	Semantic graph based	Term occurrences based
Medline data	$\surd$	
IMDB data		$\surd$
Amazon data		$\surd$
Ohsumed data		$\surd$

The semantic graph based representation was first proposed in Jiang et al. [2010a]. It serves to capture a range of document aspects: (i) word stem, (ii) word Part of Speech (POS), (iii) word order, (iv) word hypernyms, (v) sentence structures, (vi) sentence division, and (vii) sentence order. An example is given in Figure 3.4 using the first six words in a well known English sentence "The quick brown fox jumped over the lazy dog". With reference to the figure, and according to Jiang et al. [2010a], there are four different types of vertexes in this graph representation:

(1) *Structural*: Vertexes that represent sentences (S) and their component structures of noun (NP), verb (VP) and prepositional (PP) phrases. (Triangles.)

(2) *Part Of Speech (POS)*: Vertexes that represent the Part Of Speech (POS) tags of words; for example, Determiner (DT), Adjective (JJ) and Noun (NN). (Circles.)

(3) *Token*: Vertexes that represent the actual word tokens in the text. (Rectangles.)

(4) *Semantic*: Vertexes that represent additional information about the word such as its linguistic stem and other broader concepts. (Ovals.)

Each vertex also has a unique identifier and a label. There are then five different types of edges:

[4] `http://www.ncbi.nlm.nih.gov/sites/entrez`
[5] `http://www.imdb.com`
[6] `http://www.amazon.com`
[7] `http://ir.ohsu.edu/ohsumed/ohsumed.html`

(1) *hasChild*: Edges which record the structure of the text such as a sentence having a noun phrase and a verb phrase or a noun phrase containing an adjective.

(2) *isToken*: Edges which link the part of speech of a token to the token itself.

(3) *next*: Edges which record the order of the words and sentences in the text.

(4) *stem*: Edges which link to the linguistic stem of the word.

(5) *hyp*: Edges which link to a broader concept.

With respect to the example given in Figure 3.4 the vertexes are connected by "next" edges. Employing the above graph representation means that each sentence, in each document, is encoded and linked together to form one graph per document.

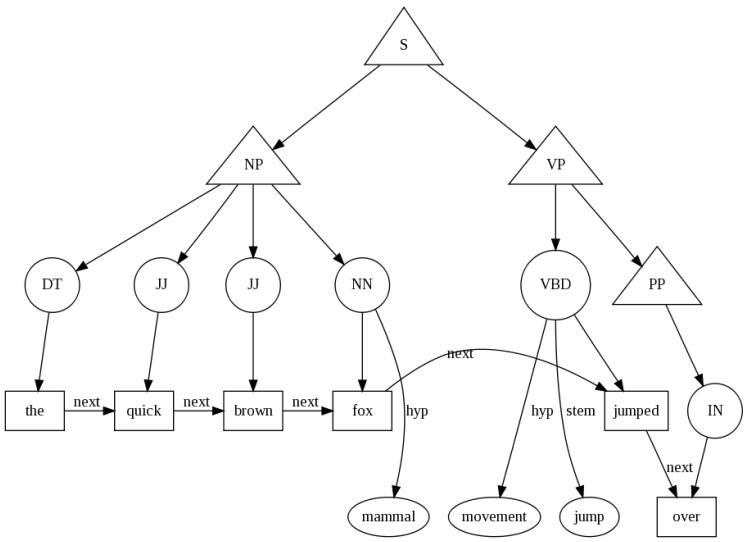


Figure 3.4: An example of semantic graph based representation

Using the above, a set of documents were extracted from the Medline collection according to their Medical Subject Heading (MeSH) fields so that a two class ("polymerase chain reaction" and "magnetic resonance imaging") data set was produced. Every document was divided into sentences using a regular expression based tokenizer and then each sentence was POS tagged, using Tsuruoka and Tsujii's "geniatagger" [Tsuruoka and Tsujii, 2005], producing a sequence of "word/POS" tokens plus the lemma (stemmed form) of each word. This tagged output was then fed into a structural parser which produces a tree with noun, verb and prepositional phrases. The nouns and verbs were then "looked up" in the WordNet thesaurus[8] and up to five broader terms added into the tree to form a better text representation and consequently classification accuracy. Every vertex in the tree was also weighted using domain knowledge: the structural vertexes were assigned a static low weight of 1, the POS vertexes were assigned a static

[8] `http://wordnet.princeton.edu/`

weight of 10, token vertexes were weighted according to their frequency in the data set (using a $TF \cdot IDF$ method), stems were assigned a weighting of half the value of the token and hypernyms a quarter value of the token. The characteristics of the resulting tree data set are presented in Table 3.7.

Table 3.7: Properties of tree represented Medline documents

	RT3
# Trees	200
Max # edges	3002
Average # edges	1141
Max # vertexes	3003
Average # vertexes	1142
# Vertex labels	10069
# Edge labels	6

3.2.4 RG1 - Chemical compound analysis scenario

The second groups of real application data sets (RG1) were drawn from the domain of chemical compound analysis. In this scenario two collections of chemical compound data, available from the Developmental Therapeutics Program (DTP) at the National Cancer Institute[9], were used:

- **CH1 - AIDS Antiviral Screening Data**: This data set consisted of 42867 chemical compounds screened for anti-HIV activity[10]. Full details can be found in Weislow et al. [1989]. The compounds are classified into three classes. Among them, 41179 belong to CI (Confirmed Inactive), 1081 belong to CM (Confirmed Moderately active) and 422 belong to CA (Confirmed Active). It should be noted here that this data set has, and continues to be, widely used for testing a variety of frequent subgraph mining algorithms [Borgelt and Berthold, 2002, Yan and Han, 2003, Nijssen and Kok, 2004, Deshpande et al., 2005, Fatta and Berthold, 2005].

- **CH2 - Human Tumour Cell Line Screen Data**: This data set comprises 42247 chemical compounds screened for evidence for inhibiting the development of human tumour cells[11]. The data set contains no class labels and thus can not be used for supervised learning purposes (but can be used for frequent subgraph mining).

Both the CH1 and the CH2 data sets are stored in MDL SD format[12]. Therefore, a MDL SD parser was written by the author to construct data sets featuring one

[9] http://dtp.nci.nih.gov/docs/3d_database/Structural_information/structural_data.html
[10] http://dtp.nci.nih.gov/docs/aids/aids_data.html
[11] http://dtp.nci.nih.gov/docs/cancer/cancer_data.html
[12] http://www.mdli.com/downloads/public/ctfile/ctfile.jsp

graph transaction per chemical compound; where vertexes represented atoms and edges bonds between atoms. Each vertex included a vertex label describing the nature of the atom type and each edge a edge label giving the *bond type*. The characteristics of the extracted graph represented chemical compounds data sets are described in Table 3.8.

Table 3.8: Properties of chemical compound graph data sets

	CH1	CH2
# graphs	42682	42247
Max # edges	441	229
Average # edges	47	28
Max # vertexes	438	223
Average # vertexes	45	26
# Vertex labels	63	67
# Edge labels	3	3

3.2.5 RG2 - Mammographic images represented by ARGs

For the second group of image mining data sets the mammographic image database[13] [Suckling et al., 1994] collated by the Mammographic Image Analysis Society (MIAS) was used. The database contains 322, 1024×1024 pixel, digital images. The database also includes radiologists' labels on the locations of any abnormalities that may be present in each image. As indicated in Suckling et al. [1994], seven classes of abnormality are present: (i) CALC (Calcification), (ii) CIRC (Circumscribed masses), (iii) SPIC (Spiculated masses), (iv) MISC (Other), (v) ARCH (Architectural distortion), (vi) ASYM (Asymmetry), (vii) NORM (Normal). The severity of abnormality is classified into two classes (Benign vs. Malignant).

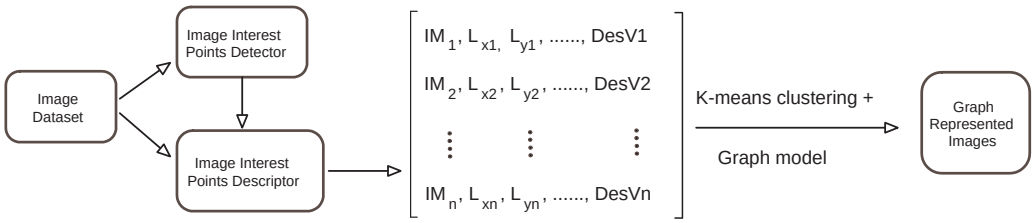
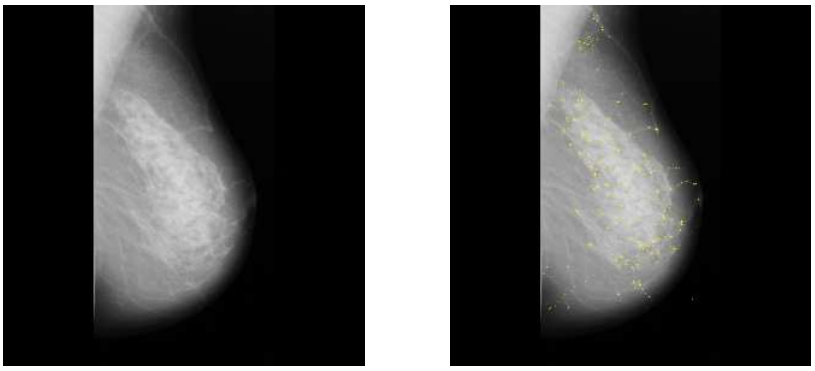


Figure 3.5: The procedure for building image interest points based graphs.

The general process of construction of these images into graphs is illustrated in Figure 3.5. Firstly, an image interest points detector [Tuytelaars and Mikolajczyk, 2008] is applied to the images and a set of image *interest points* computed using an image interest points descriptor such as SIFT [Lowe, 2004]. An example of the image interest points identified by a Harris-Affine detector [Mikolajczyk and Schmid, 2002] on one mammographic image is shown in Figure 3.6. Secondly, after obtaining the

[13] `http://peipa.essex.ac.uk/info/mias.html`

interest points, a K-means clustering is applied to these points to identify K clusters, and consequently each interest point is encoded using the index of the cluster where it belong. Thus, each cluster can be considered as a *visual word* representing a unique image feature shared by the interest points in that cluster. Thus the clustering is a process for building a vocabulary of visual words such that the number of clusters K determines the size of the vocabulary. Generally, K varies from hundreds to over tens of thousands. Finally, a feature graph for each image is constructed using these K visual words. For each graph represented image, all the image interest points belonging to one of K clusters are considered as one vertex, a link between two vertexes is included if the similarity between two clusters (i.e. visual words) is over some threshold. In Figure 3.5 $IM_1, \cdots, IM_n$ represent image interest points generated by the interest points descriptor, $(L_{x1}, L_{y1}), \cdots, (L_{xn}, L_{yn})$ represent the row and column locations of image points, and $DesV1, \cdots, DesVn$ represent the descriptor vectors for image points.



(a) Raw mammographic image

(b) Mammographic image with interest points

Figure 3.6: An example of interest points identified by a Harris-Affine detector. The yellow points on (b) denote the rich information obtained by the detector on (a).

The similarity between two cluster V_1 and V_2 is defined by the average similarity between their respective interest points:

$$Sim(V_1, V_2) = \frac{1}{|V_1| \times |V_2|} \sum_{ip_1 \in V_1} \sum_{ip_2 \in V_2} similarity(ip_1, ip_2) \tag{3.1}$$

where ip_1 and ip_2 denote the image interest points which belong to their respective clusters, and the similarity between two interest points is measured by a correlation (such as Pearson's Correlation) between their respective descriptor vectors.

In this thesis two vocabulary sizes for visual words were used: 80 and 100; resulting in two graph represented mammographic image data sets. The characteristics of these data sets are presented in Table 3.9.

Table 3.9: Properties of graph represented mammographic images

	MAM-V80	MAM-V100
Vocabulary size	80	100
# Graphs	322	322
Max # edges	2481	3894
Average # edges	765	1230
Max # vertexes	80	100
Average # vertexes	78	96
# Vertex labels	80	100
# Edge labels	1	1

3.2.6 RG3 - Photographic images represented by ARGs

The RG3 group of data sets was generated from the Caltech 101 image collection[14] created by Fei-Fei et al. [2004]. It comprised 170 real-world object images equally divided into two classes, 'Bonsai' and 'Sunflower' (85 images per class). An example of two images and their identified interest points is shown in Figure 3.7. Using the same procedure as that described in Figure 3.5, these images were represented as feature graphs. By varying the size of the vocabulary of visual words (500 and 1000), a sequence of two graph sets were constructed. The characteristics of these graph sets are presented in Table 3.10.

Table 3.10: Properties of RG3 data using different vocabularies

	BS-V500	BS-V1000
Vocabulary size	500	1000
# Graphs	170	170
Max # edges	1550	1370
Average # edges	443	370
Max # vertexes	218	217
Average # vertexes	107	86
# Vertex labels	499	991
# Edge labels	1	1

3.2.7 RG4 - Text mining founded on term occurrence

Inspired by the "Bag-of-Words" (BoW) representation [Salton et al., 1975], frequently used in text mining and information retrieval, the term occurrences-based graph representation captured both term occurrences and term order in texts. The basic idea of the graph representation is that, using the BoW to represent keywords (i.e. terms) identified in the document collection, if two terms co-occur within the same window (i.e. sentence, paragraph) then there is a relation (edge) between them. The process for generating graphs from documents, in this manner, is depicted in Figure 3.8. As

[14] `http://www.vision.caltech.edu/Image_Datasets/Caltech101/`

(a) Raw 'Bonsai' image

(b) 'Bonsai' image with identified interest points

(c) Raw 'Sunflower' image

(d) 'Sunflower' image with identified interest points

Figure 3.7: A second example of images and their interest points. The raw images belong to two different classes: 'Bonsai' and 'Sunflower'. Interests points, identified by a Harris-Affine detector, are shown in (b) and (d) with yellow points.

can be seen from the figure: each graph represents a document; the vertexes represent terms weighted by the term frequency (the number of times the term occurs in the document); and the edges indicate relationships between terms, when they occur together within a sentence, weighted by the strength of the relation (calculated using, for example, cosine similarity). In the figure the set {Doc-1, Doc-2, $\cdots$, Doc-n} denotes a set of documents. The labels attached to the 'Doc-1' graph vertexes, $\{f1, f2, \cdots, f6\}$ denote the keywords (terms) found in 'Doc-1'; and $(w1, w2, \cdots, w6)$ denote the weights associated with these terms. The weightings were calculated using the cosine similarity function. The 'Doc-1' graph in Figure 3.8 records the semantics of a document using an undirected graph. In some cases, where the order of the sequence of words may be important, a similar representation using a directed graph is also possible. The cosine similarity measure between two terms f_1 and f_2, employed in this thesis is given in Equation 3.2.

$$cosine(f_1, f_2) = \frac{|DS(f_1) \cap DS(f_2)|}{\sqrt{sup(f_1) \times sup(f_2)}} \tag{3.2}$$

where $DS(f_i)$ denotes the set of documents where term f_i appears and $sup(f_i)$ the

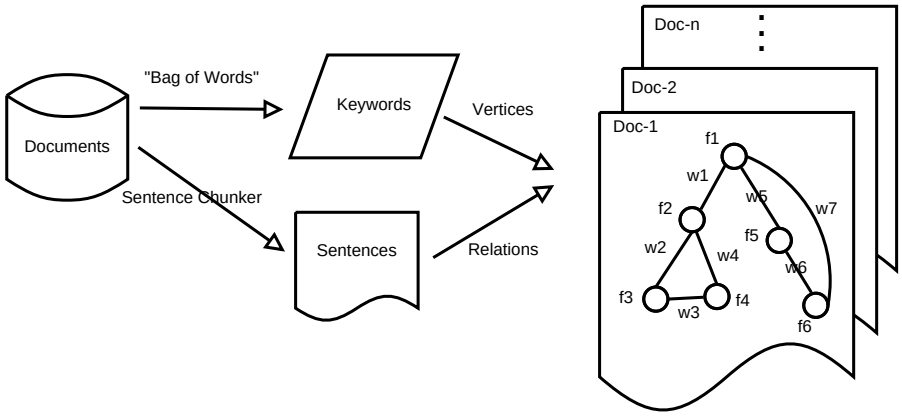


Figure 3.8: Procedure for representing documents as graphs.

number of times term f_i appears in the data set.

According to Jin and Srihari [2007], the strength of the relation between two terms (f_1 and f_2), $w(f_1, f_2)$, can also be calculated as:

$$w(f_i, f_j) = \log\left(1 + \frac{2 \cdot occ(f_i, f_j)}{occ(f_i) + occ(f_j)}\right) \tag{3.3}$$

where $occ(f_i)$ denotes the number of times term f_i appears in the document, and $occ(f_i, f_j)$ denotes the number of times term f_i and f_j co-occur within the window (e.g. sentence, paragraph).

Three different text collections[15] were selected to construct three term occurrences based graph data sets. The first collection [Ifrim et al., 2008] consisted of a set of IMDB movie plot descriptions. Each plot description comprised one or two paragraphs. The data set contained 7438 documents classified into two equal classes: Crime and Drama. The second collection comprised a selection of book reviews collated from the Amazon on-line shopping web site. This data set [Ifrim and Weikum, 2006] contained 4305 book reviews divided into two classes: Biology and Mathematics. The third collection comprised a subset of the Ohsumed collection, and consisted of a set of medical abstracts organized into Medical Subject Heading (MeSH) categories. This dataset contained 3295 documents classified into two classes: Musculoskeletal Diseases and Skin and Connective Tissue Diseases. The characteristics of the term occurrence based graph data sets built from these document collections are presented in Table 3.11.

3.2.8 RG5 - Social network mining scenario

Social network mining is a popular application domain for graph mining. The social network used with respect to the evaluation described in this thesis was extracted from

[15] Available at `http://www.birc.dk/~ifrim/`

Table 3.11: Properties of graph data sets defined using the term occurrences based representation

	IMDB	Amazon	Ohsumed
# Graphs	7438	4305	3295
Max # edges	493	2048	1080
Average # edges	32	129	141
Max # vertexes	449	225	172
Average # vertexes	34	64	55
# Vertex labels	7947	6954	5454
# Edge labels	6	6	6
Directed?	Yes	Yes	Yes

the Cattle Tracking System (CTS) database in operation in Great Britain (GB). The CTS database is maintained by the Department for the Environment, Food and Rural Affairs (DEFRA) as part of the Rapid Analysis and Detection of Animal-Related Risks (RADAR) initiative[16]. This database records all cattle movements in GB, each record describes the movement of a single animal, identified by a unique ID number, between two *holding locations* (e.g. agriculture holdings, markets, slaughter houses). Each record also includes information such as sender and receiver location and type, and animal information such as gender and breed. A social network was extracted from this database such that each vertex represented a geographical location and each edge the number of animals moved between locations (the edges are directed according to the direction of the cattle movement). Edges are annotated with a label indicating the type of movement (e.g. farmToFarm, farmToMarket, etc), and a weight indicating the number of animals moved. By utilizing the time stamp associated with movements, temporal sequences of networks were extracted describing what was conceived of as a longitudinal social network. The network comprised a sequence of graphs $\{G_1, G_2, \cdots, G_T\}$, such that G_t is the graph corresponding to the social network at time $t \in [1, T]$. The complete set of entities for the network is denoted by $V = \bigcup_{t=1}^{T} V(G_t)$ where each entity $v \in V$ is uniquely labelled, and each v can appear only once at each time step.

The longitudinal social network referenced in this thesis was collated using CTS data from 1 January 2005 to 31 December 2006. The CTS data was divided into 7-day "episodes" due to a 6-day movement restriction [Robinson and Christley, 2006] that applies to farms in GB. Figure 3.9 illustrates the process of constructing graphs from the CTS database. Firstly the selected subset of the CTS database was divided into n sub-tables according to time stamp. Each sub-table consisted of a set of records (r1,r2,$\cdots$,rn), and each record contained a set of attributes (ai1, ai2, $\cdots$). Secondly, for each sub-table, a corresponding graph describing farms and the movement between farms was constructed. Each graph included directed edges linking pairs of vertexes

[16] `http://www.defra.gov.uk/foodfarm/farmanimal/diseases/vetsurveillance/radar`

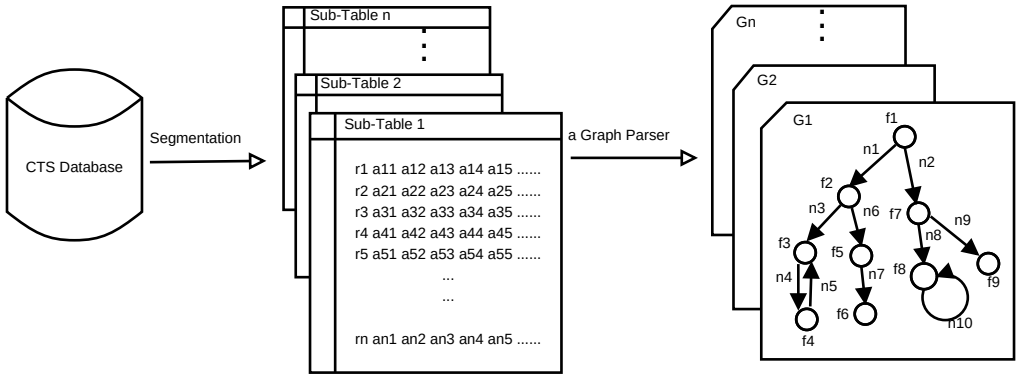


Figure 3.9: The process of building graphs from the CTS database

(indicating movement between the two vertex represented locations). In some cases several edges might connect a single pair of vertexes where (say) different breeds had been moved between location. The graphs also included *self-cycles*. In Figure 3.9 the symbols "f1, f2, $\cdots$, f10" indicate holding areas (such as farms) and the symbols "n1, n2, $\cdots$, n10" indicate the number of cattle movements. According to the research work described in Jiang et al. [2010b], three longitudinal social network data sets were extracted from the CTS database covering three different areas: (i) Lancashire[17], (ii) Scotland, (iii) Great Britain (GB). Note that the GB data set was significantly larger than the Scotland data set, which in turn was larger than the Lancashire data set. Some properties of these graphs are given in Table 3.12.

Table 3.12: Properties of graphs generated using the CTS database

	Lancashire	Scotland	GB
# graphs	105	105	105
Max # edges	377	1450	9630
Average # edges	261	890	6754
Max # nodes	393	1448	7360
Average # nodes	289	933	9957
Node label count	2148	8619	62626
Edge label count	10	17	29
Self-cycle?	yes	yes	yes
Multiple-edges?	yes	yes	yes

3.3 Summary of Data

From the foregoing it can be seen that a variety of tree and graph representations feature in the data sets used to evaluate the algorithms proposed in this thesis. Some of the representations are tree representation, some of them are undirected graphs and

[17] Lancashire is a county in the North West of England

Table 3.13: Summary of the graphic data employed in this thesis

Data Type	Trees	Undirected Graphs	Directed Graphs	Weighted	Class Labels
ST1	√				
ST2	√				√
RT1:CSLOGS-ALL	√				
RT1:CSLOGS-1 & 2	√				√
RT2	√				√
RT3	√			√	√
RG1:CH1		√			√
RG1:CH2		√			
RG2		√		√	√
RG3		√		√	√
RG4			√	√	√
RG5			√	√	√

some are directed graphs; some also have predefined weightings. Table 3.13 gives a summary of the data referenced in the the thesis. In the table the 'Data Type' column indicates different types of data, the following three columns indicate the nature of the representation, the second to last column indicates whether the data includes predefined weights, and the last column indicates whether the data includes predefined class labels.

Two graph file formats were used: (i) GraphML, and (ii) simple LineGraph. The first format was the preferred format used in the author's research group and the second is widely used by researchers in the chemical informatics domain. For ease of comparison of different algorithms, a special simple LineGraph parser was created, by the author, to handle graphs using the simple LineGraph format. A description of these two graph file formats is provided in Appendix A.1 and A.2 respectively.

Chapter 4

Weighting Functions for Vertexes and Edges in Graphs

In this chapter, a sequence of approaches to determining and applying weights to either vertexes or edges is discussed. Practically, it is not common to assign weights to both vertexes and edges. Therefore, approaches to determining weights to both vertexes and edges are ignored in this thesis. The way in which these weighting functions may be employed to achieve the desired weighted frequent subgraph mining (i.e. how they can be incorporated into *weighting schemes* is described in Chapters 5 and 8). According to the nature of the graph data to be mined, vertex and edge weightings can be roughly divided into two categories:

(i) **Structural weighting functions:** weighting functions that use information derived from the structure of the input graph set to determine individual vertex or edge weights.

(ii) **Content weighting functions:** weighting functions that derive weights using domain knowledge supplied with the input graph set. This domain knowledge may be in the form of class labels or user-supplied (pre-defined) vertex or edge weights.

These two types of functions are discussed in Sections 4.1 and 4.2 respectively, and a summary is presented in Section 4.3. In many cases it is not possible to obtain appropriate domain knowledge to facilitate the mining task, this is thus the motivation for the idea of structural weighting. Experiments (reported in Chapter 6) indicate that where domain knowledge is available better results are produced using content weighting than structural weighting. Thus, structural weighting is only applicable when domain knowledge is not available. This is also why the combination of structural and content weighting functions are not considered in this thesis.

4.1 Structural Weighting Function

Structural weighting functions derive weights for vertexes or edges purely from the "structure" of the input graph set. More specifically the number of times of vertexes or edges that occur in the graph set is utilized to compute weights. In this section, a number of structural weighting methods are considered; some can be applied to both vertexes and edges; others can only be applied to edges. Five methods for calculating structural weightings are introduced:

(1) SW1 - Normalized occurrences based method

(2) SW2 - Phi correlation coefficient based method

(3) SW3 - Normalized mutual information based method

(4) SW4 - Mutual information based method

(5) SW5 - Point-wise mutual information based method

Each one of these methods will be discussed in the following sub-sections. Note that the first is applicable to the calculation of both vertex and edge weights, the last is applicable to the calculation of vertex weights, and the remainder can only be used to determine edge weightings.

4.1.1 SW1 - Normalized occurrences based method

The normalized occurrences based method, introduced by the author in Jiang and Coenen [2008], can be used to assign weights to either vertexes or edges. In this sub-section, the method is illustrated in the context of assigning weights to edges, a very similar process can be followed to assigning weights to vertexes.

Definition 4.1.1. *Let $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$ be a graph database, and e_i be an edge of a subgraph g, then the weight of e_i with regard to $\mathbb{GD}$, is defined as:*

$$w_{\mathbb{GD}}(e_i) = \frac{occ(e_i)}{\sum_{1 \leq i \leq n} size(G_i)} \quad . \tag{4.1}$$

where $occ(e_i)$ denotes the number of times of e_i occurs in $\mathbb{GD}$, and $size(G_i)$ denotes the size of G_i in terms of the number of edges in G_i.

Example: Considering a graph database $\mathbb{GD} = \{G1, G2\}$ as shown in Figure 4.1 (note that for ease of illustration the symbol next to each edge indicates the edge label, vertex labels are not included). Given an edge with a label 'a' in the candidate subgraph g (in Figure 4.1), $occ(a) = 3$, and $\sum_{1 \leq i \leq 2} size(Gi) = 10$. Thus, $w_{\mathbb{GD}}(a) = 3/10 = 0.3$. Similarly, for an edge with a label 'e' in g, $w_{\mathbb{GD}}(e) = 1/10 = 0.1$.

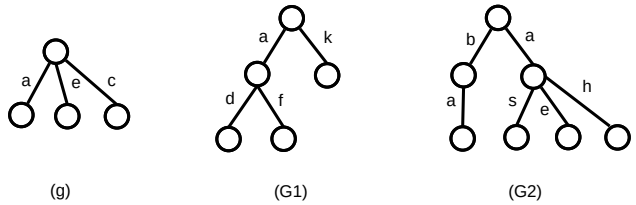


Figure 4.1: Normalized occurrences based method example

4.1.2 SW2 - Phi correlation coefficient based method

The phi correlation coefficient based method described by the author in Jiang et al. [2009] is directed at generating edge weightings only. The phi correlation coefficient is a measure of the strength of association between two binary variables [Reynolds, 1977]. In fact, it is a simplified form of Pearson's correlation coefficient for binary variables. As the name suggests the SW2 method adopts phi correlation coefficient (PCC) to determine the weights that may be allocated to edges. If two vertexes forming a 1-edge subgraph are considered as two variables, then PCC can be used to compute the correlation between them, producing a value between -1 and $+1$ inclusive. A value of 1 implies a perfect positive relationship, a value of -1 implies a perfect negative relationship, and the closer the phi coefficient is to either 1 or -1, the stronger the relationship between the variables. If the variables are independent, the phi coefficient is 0. Thus, PCC, which is in spirit similar to Xiong et al. [2004], can be expressed as:

Definition 4.1.2. *Given a graph database $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, and one edge e_i connecting two vertexes v_A and v_B that belong to a subgraph g, let the number of graph transactions where v_A and v_B occur equal $occ(v_A)$ and $occ(v_B)$ respectively, and the number of graph transactions where v_A and v_B co-occur equal $occ(v_A, v_B)$. Then, the weight of e_i, with regard to $\mathbb{GD}$, can be defined as*

$$w_{\mathbb{GD}}(e_i) = \phi(v_A, v_B) = \frac{sup(v_A, v_B) - sup(v_A) \cdot sup(v_B)}{\sqrt{sup(v_A) \cdot sup(v_B) \cdot (1 - sup(v_A)) \cdot (1 - sup(v_B))}} \quad . \tag{4.2}$$

Where $sup(v_A, v_B) = \frac{occ(v_A, v_B)}{|\mathbb{GD}|}$, $sup(v_A) = \frac{occ(v_A)}{|\mathbb{GD}|}$, and $sup(v_B) = \frac{occ(v_B)}{|\mathbb{GD}|}$. Note if $occ(v_A) = 0$ or $occ(v_A) = |\mathbb{GD}|$ ($occ(v_B) = 0$ or $occ(v_B) = |\mathbb{GD}|$) then $w_{\mathbb{GD}}(e_i) = 0$. In addition, if a negative value of (4.2) is encountered, an absolute value of it is used instead, because a negative coefficient is treated as important as a positive coefficient for the SW2 method.

Inspired by Xiong et al. [2004], if $occ(v_A) \geq occ(v_B)$, the upper bound of the $w_{\mathbb{GD}}(e_i)$ is:

$$ubound(w_{\mathbb{GD}}(e_i)) = \sqrt{\frac{occ(v_B)}{occ(v_A)}} * \sqrt{\frac{|\mathbb{GD}| - occ(v_A)}{|\mathbb{GD}| - occ(v_B)}} \quad . \tag{4.3}$$

As can be seen in (4.3), $ubound(w_{\mathbb{GD}}(e_i))$ only relies on the value of $occ(v_A)$ and $occ(v_B)$, since $|\mathbb{GD}|$ is inferred from the input. Thus, if a user requires that the weight of every edge e_i in a subgraph g has to be greater than and equal to some threshold, the upper bound of the weight of e_i can be utilized to filter those edges that do not satisfy the threshold without recourse to computing the value of $occ(v_A, v_B)$.

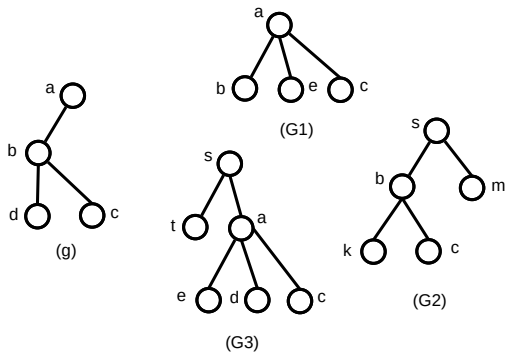


Figure 4.2: PCC based method example

Example: Considering the graph database, $\mathbb{GD} = \{G_1, G_2, G_3\}$ as shown in Figure 4.2 (again for ease of illustration the symbol next to each vertex indicates the vertex label, edge labels are not included). Given an edge (a,b) in the candidate subgraph g (in Figure 4.2), $occ(a, b) = 1$, $occ(a) = 2$, and $occ(b) = 2$. Thus, $w_{\mathbb{GD}}(a, b) = \frac{1/3-2/3\times2/3}{\sqrt{2/3\times2/3\times1/3\times1/3}} = -0.5$. Similarly, for an edge (b,d), $w_{\mathbb{GD}}(b, d) = \frac{0-2/3\times1/3}{\sqrt{2/3\times2/3\times1/3\times1/3}} = -1$.

4.1.3 SW3 - Normalized mutual information based method

The normalized mutual information (NMI) metric [Ke et al., 2008] was designed to capture the dependence among two connecting vertexes. According to Cover and Thomas [1991], the mutual information between two discrete random variables X and Y is defined as:

$$MI(X, Y) = \sum_{x \in X} \sum_{y \in Y} p(x, y) \log_2 \left(\frac{p(x, y)}{p(x) \cdot p(y)} \right) \quad . \tag{4.4}$$

Note that the $MI(X, Y) \geq 0$ ($MI(X, Y) == 0$, if and only if X and Y are independent).

Definition 4.1.3. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$ and a subgraph g with edges $E(g) = \{e_1, e_2, \cdots, e_k\}$, for each $e_i \in g$, let two vertexes connecting e_i be v_1 and v_2; let $A = \{v_1, \bar{v}_1\}$ where v_1 and $\bar{v}_1$ denote that v_1 occurs and does not occur in $\mathbb{GD}$ respectively, and let $B = \{v_2, \bar{v}_2\}$ where v_2 and $\bar{v}_2$ denote that v_2 occurs and does not occur in $\mathbb{GD}$ respectively. Thus, according to (4.4), the mutual information between A and B, denoted as $MI(A, B)$, is defined as:*

$$
\begin{aligned}
MI(A,B) &= \textstyle\sum_{a\in A}\sum_{b\in B} p(a,b)\log_2\frac{p(a,b)}{p(a)\cdot p(b)} \\
&= p(v_1\wedge v_2)\times\log_2\tfrac{p(v_1\wedge v_2)}{p(v_1)\times p(v_2)} + p(v_1\wedge\bar{v}_2)\times\log_2\tfrac{p(v_1\wedge\bar{v}_2)}{p(v_1)\times p(\bar{v}_2)} + \\
&\quad p(\bar{v}_1\wedge v_2)\times\log_2\tfrac{p(\bar{v}_1\wedge v_2)}{p(\bar{v}_1)\times p(v_2)} + p(\bar{v}_1\wedge\bar{v}_2)\times\log_2\tfrac{p(\bar{v}_1\wedge\bar{v}_2)}{p(\bar{v}_1)\times p(\bar{v}_2)} \ .
\end{aligned} \tag{4.5}
$$

where $p(v_1 \wedge v_2)$ denotes the probability of v_1 and v_2 that co-occur in $\mathbb{GD}$, $p(v_1 \wedge \bar{v}_2)$ denotes the probability of the co-occurrences of v_1 and $\bar{v}_2$ in $\mathbb{GD}$, $p(\bar{v}_1 \wedge v_2)$ denotes the probability of the co-occurrences of $\bar{v}_1$ and v_2 in $\mathbb{GD}$, and $p(\bar{v}_1 \wedge \bar{v}_2)$ denotes the probability of both v_1 and v_2 that do not occur in $\mathbb{GD}$. Inspired by the work described in Ke et al. [2008], the weight of e_i, which is computed by the NMI between A and B, is further defined as:

$$
w_{\mathbb{GD}}(e_i) = NMI(A,B) = \frac{MI(A,B)}{maximum\{MI(A,A), MI(B,B)\}} \ . \tag{4.6}
$$

where $MI(A,A) = entropy(A) = -\sum_{a\in A} p(a)\log_2 p(a) = -(p(v_1)\times\log_2 p(v_1) + p(\bar{v}_1)\times\log_2 p(\bar{v}_1))$ and $MI(B,B) = entropy(B) = -\sum_{b\in B} p(b)\log_2 p(b) = -(p(v_2)\times\log_2 p(v_2) + p(\bar{v}_2)\times\log_2 p(\bar{v}_2))$.

According to Cover and Thomas [1991], $MI(A,B)$ features two properties:

(i) $MI(A,B) \geq 0$,

(ii) $MI(A,B) \leq entropy(A)$ and $MI(A,B) \leq entropy(B)$.

Thus,

$$
0 \leq MI(A,B) \leq minimum\{entropy(A), entropy(B)\} \ . \tag{4.7}
$$

Therefore, using (4.6), the lower bound and upper bound of $NMI(A,B)$ are:

$$
0 \leq NMI(A,B) \leq \frac{minimum\{entropy(A), entropy(B)\}}{maximum\{entropy(A), entropy(B)\}} \ . \tag{4.8}
$$

The upper bound of $NMI(A,B)$ can be used to shorten the computation time required for the calculation of NMI values, because the computation time required to calculate the upper bound is less than that required to calculate the exact NMI value. If the upper bound is less than some threshold, then there is no need to calculate the actual NMI value.

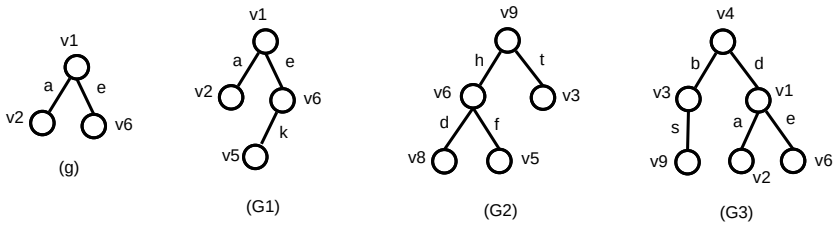


Figure 4.3: NMI based method example

Example: Considering a graph data set $\mathbb{GD} = \{G1, G2, G3\}$ as shown in Figure 4.3, where the symbol next to each edge indicates the edge label and the symbol next to each vertex indicates the vertex label. Considering the edge 'a' in the candidate subgraph g (in Figure 4.3), $p(v1) = 2/3$, $p(v2) = 2/3$, $p(\bar{v1}) = 1/3$, $p(\bar{v2}) = 1/3$, $p(v1, v2) = 2/3$, $p(v1, \bar{v2}) = 0$, $p(\bar{v1}, v2) = 0$, $p(\bar{v1}, \bar{v2}) = 1/3$.

Let $A = \{v1, \bar{v1}\}$, $entropy(A) = -(2/3 \times \log_2(2/3) + 1/3 \times \log_2(1/3)) \approx 0.9183$. Let $B = \{v2, \bar{v2}\}$, $entropy(B) = -(2/3 \times \log_2(2/3) + 1/3 \times \log_2(1/3)) \approx 0.9183$. Thus, $MI(A, B) = p(v1, v2) \times \log_2 \frac{p(v1,v2)}{p(v1)\times p(v2)} + p(v1, \bar{v2}) \times \log_2 \frac{p(v1,\bar{v2})}{p(v1)\times p(v2)} + p(\bar{v1}, v2) \times \log_2 \frac{p(\bar{v1},v2)}{p(\bar{v1})\times p(v2)} + p(\bar{v1}, \bar{v2}) \times \log_2 \frac{p(\bar{v1},\bar{v2})}{p(\bar{v1})\times p(\bar{v2})} = 2/3 \times \log_2 \frac{2/3}{2/3\times 2/3} + 1/3 \times \log_2 \frac{1/3}{1/3\times 1/3} \approx 0.9183$, and $NMI(A, B) = 0.9183/0.9183 = 1$. Thus, the weight of edge a is $w_{\mathbb{GD}}(a) = 1$.

Similarly, for edge 'e' in g, let $A = \{v1, \bar{v1}\}$, $entropy(A) = -(2/3\times\log_2(2/3)+1/3\times \log_2(1/3)) \approx 0.9183$; let $B = \{v6, \bar{v6}\}$, $entropy(B) = 0$. Thus, $MI(A, B) = p(v1, v6) \times \log_2 \frac{p(v1,v6)}{p(v1)\times p(v6)} + p(\bar{v1}, v6) \times \log_2 \frac{p(\bar{v1},v6)}{p(v1)\times p(v6)} = 2/3 \times \log_2 \frac{2/3}{2/3\times 1} + 1/3 \times \log_2 \frac{1/3}{1/3\times 1} = 0$, and $NMI(A, B) \approx 0/0.9183 = 0$. Thus, the weight of edge e is $w_{\mathbb{GD}}(e) = 0$.

4.1.4 SW4 - Mutual information based method

The mutual information based structural weighting method, as suggested by the name, makes use of the *mutual information* [Vinh et al., 2009] between two vertexes to determine the weight for the edge connecting the two vertexes. Mutual information, in this context, quantifies the degree of dependence of any two vertexes in the subgraph. It was suggested in Li et al. [2006], that mutual information reflects the "closeness" (connectedness) of two vertexes; vertexes with high mutual information tend to form a strong community. Thus, if a subgraph consists of vertexes sharing high mutual information, then it can be considered to be an important subgraph. Inspired by the work of Li et al. [2006], the mutual information between two vertexes is employed to compute the weight for the edge connecting these two vertexes.

Definition 4.1.4. *Given a graph database $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, and an edge e_i connecting two vertexes v_A and v_B that belong to a subgraph g, let the number of times of e_i that occurs in $\mathbb{GD}$ equal $occ(e_i)$ and the total number of edges in $\mathbb{GD}$ equal L; let the degrees of v_A and v_B equal $deg(v_A)$ and $deg(v_B)$ respectively and $D = \sum_{v_j \in g} deg(v_j)$. Then, the weight of e_i is calculated as:*

$$w_{\mathbb{GD}}(e_i) = p(e_i) \times \log_2 \left(\frac{p(e_i)}{p(v_A) \times p(v_B)} \right) \quad . \tag{4.9}$$

where, $p(e_i) = occ(e_i)/L$, $p(v_A) = deg(v_A)/D$, and $p(v_B) = deg(v_B)/D$.

Example: Considering the graph database $\mathbb{GD} = \{G_1, G_2\}$ shown in Figure 4.4 (as previously, for ease of illustration, the symbol next to each edge indicates the edge label, vertex labels are not included). Considering the edge 'a' in the candidate subgraph g

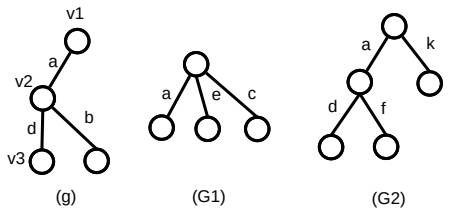


Figure 4.4: Mutual information based method example

(in Figure 4.4), $occ(a) = 2$, $deg(v1) = 1, deg(v2) = 3, D = 6$, and $L = 7$. Thus, the weight of edge a is $m_{\mathbb{GD}}(a) = 2/7 \times \log_2(\frac{2/7}{1/6 \times 3/6}) \approx 0.508$. Similarly, for an edge with a label 'd', $m_{\mathbb{GD}}(d) = 1/7 \times \log_2(\frac{1/7}{3/6 \times 1/6}) \approx 0.111$.

4.1.5 SW5 - Point-wise mutual information based method

The point-wise mutual information based method, first introduced in Jiang et al. [2010b], utilizes point-wise mutual information (PMI) [Church and Hanks, 1990] to determine the weight of a vertex.

Definition 4.1.5. *Given a graph database $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, for any single vertex v_i in a subgraph g, let the degree of v_i equal $deg(v_i)$, the set of edges incident to v_i equal $E(v_i)$. Then for each $e_k \in E(v_i)$, let the two vertexes connected by e_k be v_A and v_B, the number of graph transactions where v_A occurs equal $occ(v_A)$, the number of graph transactions where v_B occurs equal $occ(v_B)$, and the number of graph transactions where e_k occurs equal $occ(e_k)$, then the weight of v_i is defined as:*

$$w_{\mathbb{GD}}(v_i) = \begin{cases} \frac{\sum_{e_k \in E(v_i)} PMI(e_k)}{deg(v_i)-1}, & deg(v_i) \neq 1 \text{ or } 0; \\ 0, & otherwise \ . \end{cases} \tag{4.10}$$

Where $PMI(e_k)$ denotes the point-wise mutual information between v_A and v_B, which is defined as:

$$PMI(e_k) = \begin{cases} \log_2\left(\frac{p(e_k)}{p(v_A) \cdot p(v_B)}\right), & p(v_A) \neq 0 \text{ or } p(v_B) \neq 0 \text{ or } p(e_k) \neq 0; \\ 0, & otherwise \ . \end{cases} \tag{4.11}$$

where $p(e_k) = occ(e_k)/|\mathbb{GD}|$, $p(v_A) = occ(v_A)/|\mathbb{GD}|$, and $p(v_B) = occ(v_B)/|\mathbb{GD}|$.

Example: Considering a graph database $\mathbb{GD} = \{G_1, G_2, G_3\}$ as shown in Figure 4.5, where the symbol next to each edge indicates the edge label and the symbol next to each vertex indicates the vertex label. Given the vertex labelled 'v1' in the candidate subgraph g (in Figure 4.5), the two edges 'a' and 'e' are incident to 'v1', $p(a) = 2/3$, $p(e) = 3/3$, $p(v1) = 2/3$, $p(v2) = 2/3$, and $p(v6) = 2/3$. Thus, $PMI(a) = \log_2(\frac{2/3}{(2/3) \times (2/3)}) = \log_2(3/2) \approx 0.585$, $PMI(e) = \log_2(\frac{1}{(2/3) \times (2/3)}) = \log_2(9/4) \approx 1.1699$, and the weight to be attached to $v1$ is $w_{\mathbb{GD}}(v1) = PMI(a) + PMI(e) \approx 1.7549$.

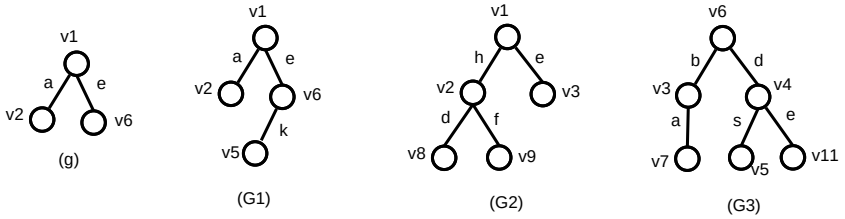


Figure 4.5: PMI based method example

4.2 Content Weighting Function

As noted in the introduction to this chapter content weighting functions use domain knowledge concerning the nature of the input graph data set. The nature of the domain knowledge can be in two forms:

User defined weights vertex or edge weights that have, by some means, been predetermined, for example by a domain expert or an end user.

Classes predefined class labels that have been attached to each graph in the input graph data set.

Methods derived from the above can be categorized approximately into the following:

(1) CW1 - User predefined weights

(2) CW2 - Calculation of edge weights using χ^2 values

(3) CW3 - Calculation of edge weights using NMI values

Each one of these will be discussed in the following three sub-sections. Note that the first is used in conjunction with user defined weights, while the last two are used in conjunction with predefined class labels.

4.2.1 CW1 - User predefined weights

User predefined (supplied) weights can be attached directly to either vertexes (denoted by CW1-N) or edges (denoted by CW1-E). Thus, any weighting schemes to calculate weights for subgraphs can employ these weights directly. For example, given a subgraph g with vertexes$\{v_1, v_2, \cdots, v_k\}$, if the corresponding weights for vertexes are $\{w_1, w_2, \cdots, w_k\}$, then one common way of computing a weight for g is $\sum_{v_i \in g} w_i$ [Chartrand and Zhang, 2004].

4.2.2 Calculation of edge weights using class labels

In real applications, user predefined weights for vertexes or edges in the graph are not always available. Under this situation, if class labels for graphs in the input graph

data set are provided, this knowledge can be used to determine the weights for edges or vertexes. There are a number of *feature selection* techniques that can be deployed for this purpose, examples including information gain, mutual information, and χ^2. In this thesis, two methods were adopted to assign weights to edges according to their association with given class labels: (i) calculation of edge weights using χ^2 values and (ii) calculation of edge weights using normalized mutual information values. Each will be examined in Sub-sections 4.2.2.1 and 4.2.2.2 below. Methods to assign weights to vertexes can be derived in a similar manner.

4.2.2.1 CW2 - Calculation of edge weights using χ^2 values

Using χ^2 values to compute edge weights was initially described by the author in Jiang et al. [2009]. Inspired by the work described in Yang and Pedersen [1997] that used the χ^2 statistic measure as a feature selection technique to facilitate text categorization in the context of frequent pattern based classification, the CW2 method adopts the pairwise χ^2 measure to capture the goodness of the edge in a graph with respect to the class label with which that graph is associated.

Table 4.1: A two-way contingency table of e_i and c_j

	c_j	$\bar{c_j}$	$\sum row$
e_i	a	b	$(a+b)$
$\bar{e_i}$	c	d	$(c+d)$
$\sum column$	$n1=(a+c)$	$n2=(b+d)$	$N=n1+n2$

Given a graph database $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, a set of class labels, $C = \bigcup_{j=1}^{k} c_j$, such that each graph G_i is associated with a single class label, and an edge e_i of a subgraph g. Then, using a 2×2 two-way contingency table of e_i and c_j as shown in Table 4.1, let a denote the number of times e_i and c_j co-occur and b the number of times e_i occurs without c_j; let c denote the number of times c_j occurs without e_i, and d the number of times neither e_i nor c_j occurs. Thus, the edge "goodness" measure of e_i with regard to c_j is defined to be the normalized deviation of observation (O_{ij}) from expection (E_{ij}); namely,

$$\chi^2(e_i, c_j) = \sum_{i \in \{0,1\}, j \in \{0,1\}} \frac{(O_{ij} - E_{ij})^2}{E_{ij}} \ . \tag{4.12}$$

According to Sheskin [1997], the χ^2 test for 2×2 table is equivalent to calculating the Z test for two independent proportions sampled from the same population. Thus, the Z test for comparing two proportions is given by

$$Z = \frac{p_1 - p_2}{\sqrt{p(1-p)\left(\frac{1}{n_1} + \frac{1}{n_2}\right)}} = \sqrt{\chi^2} \ . \tag{4.13}$$

If let $p_1 = a/(a+c)$, $p_2 = b/(b+d)$, and $p = (a+b)/N$, then the short-cut formula for $\chi^2(e_i, c_j)$ is given by:

$$\chi^2(e_i, c_j) = Z^2 = \left(\frac{\frac{a}{a+c} - \frac{b}{b+d}}{\sqrt{\frac{a+b}{N}\frac{c+d}{N}\left(\frac{1}{a+c} + \frac{1}{b+d}\right)}} \right)^2 = \frac{N(ad-bc)^2}{(a+b)(a+c)(b+d)(c+d)} \quad (4.14)$$

For each class c_j, the χ^2 statistic between e_i and c_j is computed to measure the strength of e_i with respect to c_j, and then the average χ^2 value for e_i calculated. This average value is considered to be the weight of e_i. Formally:

$$w_{\mathbb{GD}}(e_i) = \chi^2_{avg}(e_i) = \sum_{j=1}^{k} pr(c_j)\chi^2(e_i, c_j) \ . \quad (4.15)$$

where $pr(c_j)$ denotes the probability of c_j's occurrences in $\mathbb{GD}$.

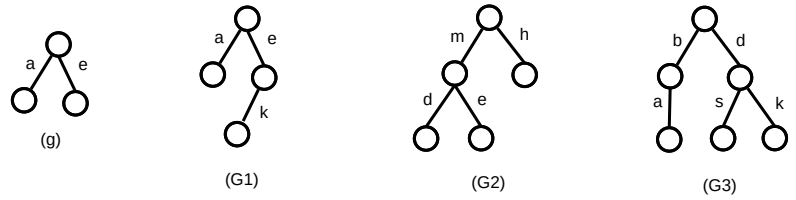


Figure 4.6: χ^2 based method example

Example: Considering the graph database $\mathbb{GD} = \{G_1, G_2, G_3\}$ as shown in Figure 4.6, (again for ease of illustration, the symbol next to each edge indicates the edge label and vertex labels are not included.) G1 and G2 belong to class 'A' and G3 belongs to class 'B'. Considering the edge 'a' in the candidate subgraph g (in Figure 4.6), the two-way contingency tables for 'a' and class labels are shown in Table 4.2.

Table 4.2: Two-way contingency tables for 'a' and class labels

(a) 'a' and 'A'

	A	$\bar{A}$	$\sum$
a	1	1	2
$\bar{a}$	1	0	1
$\sum$	2	1	

(b) 'a' and 'B'

	B	$\bar{B}$	$\sum$
a	1	1	2
$\bar{a}$	0	1	1
$\sum$	1	2	

$\chi^2(a, A) = \frac{3\times(-1)^2}{2\times1\times2\times1} = 3/4$, $\chi^2(a, B) = \frac{3\times1}{2\times1\times2\times1} = 3/4$, and $pr(A) = 2/3, pr(B) = 1/3$. Thus, the weighting for edge a is given by $w_{\mathbb{GD}}(a) = 2/3 \times (3/4) + 1/3 \times (3/4) = 3/4 = 0.75$.

Similarly, considering the edge 'e' in g, the contingency tables for 'e' and class labels are shown in Table 4.3.

Table 4.3: Two-way contingency tables for 'e' and class labels

(a) 'e' and 'A'

	A	$\bar{A}$	$\sum$
e	2	0	2
$\bar{e}$	0	1	1
$\sum$	2	1	

(b) 'e' and 'B'

	B	$\bar{B}$	$\sum$
e	0	2	2
$\bar{e}$	1	0	1
$\sum$	1	2	

$\chi^2(e, A) = \frac{3\times4}{2\times1\times2\times1} = 3$, $\chi^2(e, B) = \frac{3\times(-2)^2}{1\times2\times2\times1} = 3$, and $pr(A) = 2/3, pr(B) = 1/3$. Thus, $w_{\mathbb{GD}}(e) = 2/3 \times 3 + 1/3 \times 3 = 3$.

4.2.2.2 CW3 - Calculation of edge weights using NMI values

Inspired by Xu et al. [2005], the CW3 method adopts the NMI value to quantify the global weight of the edge. This method is very similar to the SW3 method introduced in Sub-section 4.1.3. SW3 employs the NMI value that exists between two vertexes to calculate the edge weight, while CW3 employs the NMI value between an edge and class labels to calculate the edge weight.

Definition 4.2.1. *Given a graph database $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, a set of class labels $C = \bigcup_{j=1}^{k} c_j$, such that each graph G_i is associated with a single class label, and an edge e_i of a subgraph g, let $S = \{e_i, \bar{e}_i\}$ where e_i and $\bar{e}_i$ denote that e_i occurs and does not occur in $\mathbb{GD}$ respectively. Thus, according to (4.4), the mutual information between S and C is defined as:*

$$MI(S, C) = \sum_{s\in S}\sum_{c_j\in C} p(s, c_j)\log_2\left(\frac{p(s, c_j)}{p(s)\cdot p(c_j)}\right) \quad . \tag{4.16}$$

This can be further expanded as:

$$MI(S, C) = \sum_{j=1}^{k} p(e_i \wedge c_j)\log_2\left(\frac{p(e_i \wedge c_j)}{p(e_i)\times p(c_j)}\right) + \sum_{j=1}^{k} p(\bar{e}_i \wedge c_j)\log_2\left(\frac{p(\bar{e}_i \wedge c_j)}{p(\bar{e}_i)\times p(c_j)}\right) \quad , \tag{4.17}$$

where $p(e_i \wedge c_j)$ and $p(\bar{e}_i \wedge c_j)$ denote respectively the probability of e_i and c_j co-occur in $\mathbb{GD}$ and the probability of the co-occurrences of $\bar{e}_i$ and c_j in $\mathbb{GD}$; $p(e_i)$ and $p(c_j)$ denote respectively the probability of e_i's occurrences in $\mathbb{GD}$ and the probability of c_j's occurrences in $\mathbb{GD}$. If $p(e_i \wedge c_j)$, or $p(\bar{e}_i \wedge c_j)$, or $p(e_i)$, or $p(c_j)$ equals zero, the corresponding $\log_2(\cdot) = 0$. Thus, the weight of e_i is further defined as the NMI between S and C; namely,

$$w_{\mathbb{GD}}(e_i) = NMI(S, C) = \frac{MI(S, C)}{maximum\{MI(S, S), MI(C, C)\}} \quad , \tag{4.18}$$

where $MI(S,S) = entropy(S) = -\sum_{s \in S} p(s) \cdot \log_2 p(s)$, and $MI(C,C) = entropy(C) = -\sum_{c \in C} p(c) \cdot \log_2 p(c)$. According to (4.8) introduced with respect to the SW4 method (see Sub-section 4.1.3), $0 \leq NMI(S,C) \leq 1$. Furthermore, the upper bound of $NMI(S,C)$ can be used to calculate the estimated value of the edge weight.

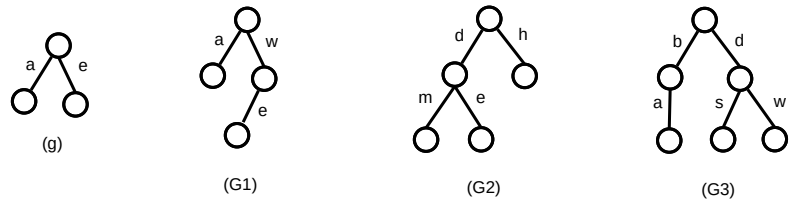


Figure 4.7: An example of computing edge weights using NMI values

Example: Considering the graph database $\mathbb{GD} = \{G_1, G_2, G_3\}$ as shown in Figure 4.7, where G1 and G2 belong to class 'A' and G3 belongs to class 'B'.

Considering the edge 'a' in the candidate subgraph g, $p(a,A) = 1/3, p(a,B) = 1/3$; $p(\bar{a},A) = 1/3, p(\bar{a},B) = 0$; $p(a) = 2/3, p(\bar{a}) = 1/3$; $p(A) = 2/3, p(B) = 1/3$. So, $MI(\{a,\bar{a}\},\{A,B\}) = 1/3 \times \log_2(\frac{1/3}{2/3 \times 2/3}) + 1/3 \times \log_2(\frac{1/3}{1/3 \times 1/3}) + 1/3 \times \log_2(\frac{1/3}{1/3 \times 2/3}) + 0 \approx 0.585$, $entropy(\{a,\bar{a}\}) = -(2/3 \times \log_2(2/3) + 1/3 \times \log_2(1/3)) \approx 0.9183$, and $entropy(\{A,B\}) = -(2/3 \times \log_2(2/3) + 1/3 \times \log_2(1/3)) \approx 0.9183$. Hence, $w_{\mathbb{GD}}(a) = NMI(\{a,\bar{a}\},\{A,B\}) \approx 0.585/0.9183 \approx 0.637$.

Similarly, for edge 'e', $p(e,A) = 2/3, p(e,B) = 0$; $p(\bar{e},A) = 0, p(\bar{e},B) = 1/3$; $p(e) = 2/3, p(\bar{e}) = 1/3$. So $MI(\{e,\bar{e}\},\{A,B\}) = 2/3 \times \log_2(\frac{2/3}{2/3 \times 2/3}) + 1/3 \times \log_2(\frac{1/3}{1/3 \times 1/3}) \approx 0.9183$, $entropy(\{e,\bar{e}\}) = -(2/3 \times \log_2(2/3) + 1/3 \times \log_2(1/3)) \approx 0.9183$. Hence, $w_{\mathbb{GD}}(e) = NMI(\{e,\bar{e}\},\{A,B\}) \approx 0.9183/0.9183 = 1$.

4.3 Summary

Two categories of weighting function: (i) structural weighting and (ii) content weighting, were introduced in this chapter. Five structural weighting functions were identified (labelled: SW1, SW2, SW3, SW4 and SW5); and three content based weighting functions were identified (labelled: CW1, CW2 and CW3). The adoption of structural weighting functions is most appropriate where no predefined weightings (or class labels) are available. The weighting functions will be used (see Chapters 5 and 8) to determine weightings to be attached to subgraphs generated during the desired weighted frequent subgraph mining process, i.e. the vertex or edge weights will be employed to compute the significance of each discovered subgraph. Five structured weighting methods were identified. Except for the normalised occurrences based method (SW1), which can be applied to both, vertexes or edges, and the point-wise mutual information based method (SW5), which can applied to vertexes only, the rest of these (SW2, SW3 and SW4) were only applicable with respect to the calculation of weightings to be attached to graph edges. The SW3, SW4, and SW5 weighting functions were all related to using

a variant of mutual information measure in different context. The SW3 method relied on the probability of the existence or absence of the vertexes to compute the edge weights, and the SW4 method utilized both the edge occurrences and the degrees of the vertexes connecting that edge to compute the edge weights. However, for SW5, both the edge occurrences and vertex occurrences, in conjunction with the degrees of the vertexes were used to compute the vertex weights. Among these three mutual information associated methods, each one of them used a slightly different form of mutual information formula, and each form of mutual information measure was employed to quantify the closeness between two vertexes (the mutual information measures were used directly to compute the edge weights in SW3 and SW4 while in SW5, the mutual information measure was used as an intermediate step to compute the vertex weights).

Content weighting functions are applicable where some prior domain knowledge (e.g. predetermined edge or vertex weightings, or graphs labelled by classes) is available. Where the weights for vertexes or edges have been predetermined, these can be used directly to determine the significance for each discovered subgraph in the mining process. In the case of graph sets supplied with class labels, such labels may be utilized to determine the strength (i.e. weight) of vertexes or edges associated with the class labels. The significance of subgraphs discovered during the mining process can then be determined using the same methods used in the case of predefined weightings.

Chapter 5

Subgraph Weighting Schemes That Maintain the DCP

In the previous chapter, Chapter 4, a number of different weighting functions for assigning weights to vertexes or edges were proposed. These functions can be incorporated into different *weighting schemes* to calculate the significance of discovered subgraphs identified as a result of frequent subgraph mining. In this chapter, we consider weighting schemes that maintain the Downward Closure Property (DCP). In the following chapter we will consider weighting schemes that adopt alternative mechanisms to limit the search space.

5.1 Overview

A fundamental characteristic of the support measure employed in transaction-based frequent subgraph mining is that: *the support of a subgraph g is always less than or equal to that of any of it's subgraphs.* This characteristic is the well-known Downward Closure Property (DCP), also referred to as anti-monotonicity: *a subgraph g is frequent if and only if all of it's subgraphs are frequent.* This property can play an important role in reducing the number of candidate patterns generated during the mining process, and consequently improve the mining performance significantly.

Thus, when considering weighting schemes, such as those advocated in this thesis, it is desirable to seek to maintain the DCP. However, as noted in Vanetik et al. [2006], it is not easy to design a subgraph weighting mechanism which maintains the DCP. For instance, given a subgraph g with assigned weights $\{w_1, w_2, \cdots, w_k\}$ for its edge set $\{e_1, e_2, \cdots, e_k\}$, a straightforward approach to calculating the weight for g can be defined to be $wt(g) = \sum_{i=1}^{k} w_i$ [Chartrand and Zhang, 2004]. If $wt(g)$ is less than a user provided threshold θ, for a supergraph h of g with one more edge e_{k+1}, its weight $wt(h) = \sum_{i=1}^{(k+1)} w_i$ is not necessarily less than θ, thus violating the DCP.

A variety of subgraph weighting schemes that maintain the DCP are described in this chapter. These schemes can be approximately divided into two categories. The first

category comprises weighting schemes that employ the weighting functions described in the previous chapter. The second category comprises weighting schemes that adopt alternative approaches that do not require recourse to such weighting functions.

The two categories are considered independently in Sections 5.2 and 5.3 respectively. Three schemes have been devised that fall into the first category:

1. Average Total Weighting (ATW)
2. Affinity Weighting (AW)
3. Correlation Measures based Weighting (CMW)

and one that falls into the second category:

4. Jaccard Similarity based Weighting (JSW)

The proposed algorithms have all been implemented by incorporating them into variants of three well known FSM algorithms, namely gSpan, FFSM and GASTON (see Subsection 2.6.1.2). In each case the implementations are fairly similar thus in this chapter only the implementation with respect to gSpan are detailed. The gSpan algorithm was selected as the "base algorithm" for all the weighting schemes described in this chapter because: (i) it is well established, (ii) it is well documented and (iii) it is easy to be modified.

5.2 Weighting Schemes Using Vertex or Edge Weights

In this section, the weighting schemes that use vertex or edge weights are described: (i) Average Total Weighting (ATW), (ii) Affinity Weighting (AW), (iii) Correlation Measures based Weighting (CMW). The ATW scheme can be applied to either vertex weighted or edge weighted graphs, whilst the AW and CMW schemes can be applied to edge weighted graphs only. Referring back to Chapter 4, content weighting functions are always preferred, if both structural and content weighting functions are available. For content weighting functions, the CW1 weighting function is suitable for use in the ATW and AW schemes; the CW2 and CW3 weighting functions are only devised to be used with the CMW scheme, because they formulate two correlation measures which are required by the CMW scheme. For structural weighting functions, the SW1, SW4, and SW5 are conceived to be used with the ATW scheme, while the AW scheme only uses the SW1 and SW4 weighting functions because the AW scheme requires edge weights; again the SW2 and SW3 weighting functions are only devised to be used with the CMW scheme, because of the nature of the CMW scheme. As will become apparent each of these weighting schemes is directed at a particular type of graph with a particular type of weighting function (although for some types of graphs and some

types of weighting functions more than one of the schemes may be applicable). Each of these three weighting schemes is discussed in detail below in Sub-sections 5.2.1, 5.2.2, and 5.2.3 respectively.

5.2.1 Average Total Weighting (ATW) scheme

In the Average Total Weighting (ATW) scheme [Jiang and Coenen, 2008], which is inspired by the work in Tao et al. [2003], given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, the weight for a subgraph g is calculated by dividing the sum of the average weights in the graphs that contain g with the sum of the average weights across the entire graph data set $\mathbb{GD}$. This scheme can accommodate both vertex and edge weighted graphs. In this sub-section the ATW scheme is described in terms of edge weighted graphs, the scheme can be applied to vertex weighted graphs in a similar manner. Thus:

Definition 5.2.1. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, if G_i is edge weighted by $\{w_1, w_2, \cdots, w_k\}$, then the average weight associated with G_i is defined as:*

$$W_{avg}(G_i) = \frac{\sum_{j=1}^{k} w_j}{k} \quad . \tag{5.1}$$

where w_j can be determined using appropriate weighting functions (such as those described in Chapter 4). The total weight of $\mathbb{GD}$ is further defined as:

$$W_{sum}(\mathbb{GD}) = \sum_{i=1}^{n} W_{avg}(G_i) \quad . \tag{5.2}$$

Using both (5.1) and (5.2), the weight of a subgraph can be calculated by (5.3).

Definition 5.2.2. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots G_n\}$ and an arbitrary subgraph g, let the set of graphs where g occurs equal $\delta_{\mathbb{GD}}(g)$. Then, the weight of g with respect to $\mathbb{GD}$ is:*

$$W_{\mathbb{GD}}(g) = \frac{\sum_{G_i \in \delta_{\mathbb{GD}}(g)} W_{avg}(G_i)}{W_{sum}(\mathbb{GD})} \quad . \tag{5.3}$$

$W_{\mathbb{GD}}(g)$ is used to quantify the actual importance of each discovered subgraph g in a graph data set. The weighted support of a subgraph g is then defined as the product of the support of g and the importance factor of g:

$$wsup_{\mathbb{GD}}(g) = W_{\mathbb{GD}}(g) \cdot sup_{\mathbb{GD}}(g) = \frac{W_{\mathbb{GD}}(g) \cdot |\delta_{\mathbb{GD}}(g)|}{|\mathbb{GD}|} \quad . \tag{5.4}$$

Definition 5.2.3. *A subgraph g is weighted frequent with respect to $\mathbb{GD}$, if $wsup_{\mathbb{GD}}(g) \geq \tau$, where $0 < \tau \leq 1$ is a weighted support threshold.*

Theorem 5.2.1. *If a weighted subgraph is infrequent, then any supergraph of this subgraph is also infrequent.*

Proof. Let g be a weighted infrequent subgraph in a graph data set $\mathbb{GD}$, then $W_{\mathbb{GD}}(g) \times sup_{\mathbb{GD}}(g) < \tau$; let h be a supergraph of g, i.e. $g \subset h$, then $sup_{\mathbb{GD}}(g) \geq sup_{\mathbb{GD}}(h)$ and $W_{\mathbb{GD}}(g) \geq W_{\mathbb{GD}}(h)$. Therefore, $W_{\mathbb{GD}}(h) \times sup_{\mathbb{GD}}(h) \leq W_{\mathbb{GD}}(g) \times sup_{\mathbb{GD}}(g) < \tau$, h is a weighted infrequent subgraph. □

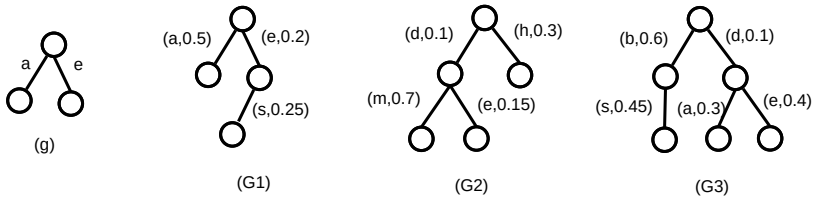


Figure 5.1: An example of calculating weights by the ATW scheme

Example: Considering the graph data set $\mathbb{GD} = \{G_1, G_2, G_3\}$ shown in Figure 5.1, where the symbol next to each edge indicates the edge label (the vertex labels are not included in the figure). For each edge label symbol '(a,b)' in the figure, a denotes the label and b denotes the weight. Given a subgraph g, which occurs in $G1$ and $G3$, $W_{sum}(\mathbb{GD}) = \frac{0.5+0.2+0.25}{3} + \frac{0.1+0.3+0.7+0.15}{4} + \frac{0.6+0.1+0.45+0.3+0.4}{5} \approx 0.992$, $W_{avg}(G3) = \frac{0.6+0.1+0.45+0.3+0.4}{5} \approx 0.3700$, $W_{avg}(G1) = \frac{0.5+0.2+0.25}{3} \approx 0.3167$. Thus, $W_{\mathbb{GD}}(g) = \frac{0.3167+0.3700}{0.992} \approx 0.6872$, $wsup_{\mathbb{GD}}(g) = 2/3 \times W_{\mathbb{GD}}(g) \approx 0.4582$.

From the above it can be easily inferred that $wsup_{\mathbb{GD}}(g)$, as defined by Equation 5.4, maintains the DCP property. Therefore, if a k-subgraph candidate is not frequent, then all of its $(k+1)$-supergraphs can be safely pruned from this branch in the lattice during the $(k+1)$ candidate generation process. It should be noted, however, that the ATW scheme will tend to bias large transaction graphs over smaller transaction graphs, thus the approach is best applied to graph sets where the individual graphs are of a similar size.

5.2.1.1 Pseudo-codes of ATW

The implementation of the ATW scheme into gSpan, to give gSpan-ATW, was based on the pseudo-codes of gSpan introduced in Section 2.10.1. Since only the procedure of 'subgSpan' as described in Section 2.10.1 needs to be modified, the pseudo-codes for integrating the ATW scheme into the revised procedure are presented in the procedure of 'subgSpan-ATW'. In the procedure, a weighted support threshold τ is introduced to replace the threshold σ used in Algorithm 2.3. Inspection of the procedure (lines 4 and 13) indicates that when the weighted support of a subgraph candidate is below some threshold, there is no need to extend that subgraph candidate, because the ATW scheme satisfied the DCP as demonstrated in Theorem 5.2.1.

Procedure subgSpan-ATW(c, $\mathbb{GD}$, τ, $\mathcal{F}$)

```
1  if c ≠ min(c) then
2  |  return
3  end
4  if W_GD(c) × sup_GD(c) ≥ τ then
5  |  F ← F ∪ {c}
6  else
7  |  return
8  end
9  C ← ∅
10 Scan GD once, find every edge e such that c can be right-most extended to c ∪ e,
   C ← c ∪ e
11 Sort C in DFS lexicographic order
12 foreach g_k ∈ C do
13 |  if W_GD(g_k) × sup_GD(g_k) ≥ τ then
14 |  |  subgSpan-ATW(g_k, GD, τ, F)
15 |  end
16 end
```

5.2.2 Affinity Weighting (AW) scheme

The Affinity Weighting (AW) scheme [Jiang et al., 2010c] is founded on two components to restrict the growth of the search space: (i) a graph distance measure, and (ii) a weighting ratio measure. For a subgraph g to be weighted frequent, both must be greater than specified user thresholds. The graph distance measure is defined as follows.

Definition 5.2.4. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, and a subgraph g, let the set of graphs where g occurs equal $\delta_{\mathbb{GD}}(g)$. Then, the weight of g is formulated as a distance metric, $dist(g, \delta_{\mathbb{GD}}(g))$:*

$$W_{\mathbb{GD}}(g) = dist(g, \delta_{\mathbb{GD}}(g)) = \frac{\sum_{G_i \in \delta_{\mathbb{GD}}(g)} 1 - \frac{|V(g) \cap V(G_i)|}{|V(g) \cup V(G_i)|}}{C} \quad . \tag{5.5}$$

which is simplified as:

$$W_{\mathbb{GD}}(g) = \frac{\sum_{G_i \in \delta_{\mathbb{GD}}(g)} 1 - \frac{|V(g)|}{|V(G_i)|}}{C} \quad . \tag{5.6}$$

Where $C = |V(g)|$ or $C = |\mathbb{GD}|$. Because of the diversity of the data sets introduced in Chapter 3, in order to obtain proper $W_{\mathbb{GD}}(g)$ values to reduce the search space, the value of C needs to be set according to the properties of the data sets. When the number of graphs in the data set is large and the size of each graph is relatively small, $C = |\mathbb{GD}|$; when the number of graphs in the data set is small and the size of each graph is very large, $C = |V(g)|$. The efficiency of applying the AW scheme to various types of data sets relies on the proper selection of C values. It should be noted that

adding vertexes to g can only reduce the value of (5.6), because $|\delta_{\mathbb{GD}}(g)|$ can not be increased. Thus the value of $dist(g, \delta_{\mathbb{GD}}(g))$ is non-increasing.

The graph distance measure is directed at the number of vertexes contained in a graph; the weighting ratio, in turn, is concerned with the edge weights. The weighting ratio of an edge-weighted subgraph g is a function $r(g)$ that returns a value between zero and one which is non-increasing in the number of edges of g. Given an edge weighted subgraph g with weights $S = \{w_1, w_2, \cdots, w_k\}$, the weighting ratio function $r(g)$ which is similar to Yun [2007], is defined as:

$$r(g) = \frac{MIN_{w_i \in S}\{w_i\}}{MAX_{w_j \in S}\{w_j\}} \quad . \tag{5.7}$$

It should be noted that w_i used in (5.7) is assumed to be positive. In this thesis, if w_i becomes negative, the absolute value of w_i is used instead.

Definition 5.2.5. *Given an edge-weighted graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, a weighted support threshold $\tau \in (0, 1]$, and a weighting ratio threshold $\lambda \in [0, 1]$, a subgraph g is weighted frequent, if the following two conditions (C1 and C2) are satisfied:*

$$\textbf{(C1)}\ wsup_{\mathbb{GD}}(g) = sup_{\mathbb{GD}}(g) \times W_{\mathbb{GD}}(g) \geq \tau, \qquad \textit{and} \qquad \textbf{(C2)}\ r(g) \geq \lambda \quad .$$

Example: Considering the graph data set $\mathbb{GD} = \{G1, G2, G3\}$ shown in Figure 5.2, where the symbol next to each edge indicates the edge label (vertex labels are not shown). In the figure the edge label symbols (a, b) associated with a subgraph g should again be interpreted as follows: a denotes the label and b denotes the weight. Given a subgraph g, which occurs in $G1$ and $G3$, $W_{\mathbb{GD}}(g) = \frac{1}{4} \times (1 - \frac{4}{4} + 1 - \frac{4}{7}) = 3/28$, $wsup_{\mathbb{GD}}(g) = 2 \times 3/28 = 0.2143$, and $r(g) = \frac{0.2}{0.5} = 0.4$.

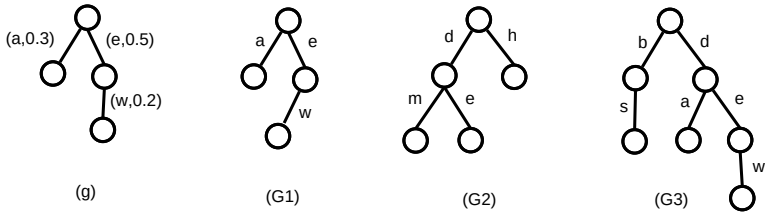


Figure 5.2: An example of calculating weights by the AW scheme

It can be inferred, from Definition 5.2.5, that both conditions maintain the DCP. Therefore these two conditions can lead to an alternative pruning strategy which may be incorporated into any standard FSM algorithms.

5.2.2.1 Pseudo-codes of AW

The gSpan algorithm was used as the base algorithm for the incorporation of the AW scheme to create gSpan-AW. The pseudo-codes for modifying the procedure of

'subgSpan' as described in Section 2.10.1 are presented in the procedure of 'subgSpan-AW'. In the procedure, except that a weighted support threshold τ and a weighting ratio threshold λ are introduced in the AW scheme, the rest of the parameters maintain the same meaning as those introduced in Algorithm 2.3. Inspection of the procedure of 'subgSpan-AW' indicates that, during the candidate generation phase, the mining will keep track of the values of both $W_{\mathbb{GD}}(g_k)$ and $r(g_k)$ (lines 4 and 13) for all candidates, and discard all those candidates that do not satisfy at least one of the conditions **(C1)** and **(C2)**.

Procedure subgSpan-AW(c, $\mathbb{GD}$, τ, λ, $\mathcal{F}$)

```
1  if c ≠ min(c) then
2  |  return
3  end
4  if W_GD(c) × sup_GD(c) ≥ τ ∧ r(c) ≥ λ then
5  |  F ← F ∪ {c}
6  else
7  |  return
8  end
9  C ← ∅
10 Scan GD once, find every edge e such that c can be right-most extended to c ∪ e,
   C ← c ∪ e
11 Sort C in DFS lexicographic order
12 foreach g_k ∈ C do
13 |  if W_GD(g_k) × sup_GD(g_k) ≥ τ ∧ r(g_k) ≥ λ then
14 |  |  subgSpan-AW(g_k, GD, τ, λ, F)
15 |  end
16 end
```

5.2.3 Correlation Measures based Weighting (CMW) scheme

In the Correlation Measures based Weighting (CMW) scheme, a range of correlation measures were employed to quantify the correlation between vertexes or edge and the classes with which the subgraph is associated. According to the availability of the class labels for graphs, the correlation measures used in the CMW scheme were divided into two categories.

- **Correlation measures that do not use class labels.** In this category, the correlation measure is designed to capture the relationship between two vertexes connected by any edge in a subgraph. Two examples are the weighting functions introduced in Section 4.1, SW2 and SW3.

- **Correlation measures do use the class labels.** In this category, the correlation measure is designed to capture the relationship between any edge in a

subgraph and the classes with which the subgraph is associated. Two examples are the weighting functions introduced in Section 4.2, CW2 and CW3.

Inspired by Ke et al. [2008], all these four weighting functions (SW2, SW3, CW2 and CW3) can be incorporated into a CMW scheme. Thus,

Definition 5.2.6. *Given a graph data set* $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$*, a support threshold* $\sigma \in (0, 1]$*, and a weighting threshold* θ*, a subgraph* g *is weighted frequent, if the following two conditions (D1 and D2) are satisfied:*

$$\textbf{(D1)}\ sup_{\mathbb{GD}}(g) \geq \sigma, \qquad \textit{and} \qquad \textbf{(D2)}\ \forall e_i \in g, \quad w_{\mathbb{GD}}(e_i) \geq \theta \quad .$$

Where $w_{\mathbb{GD}}(e_i)$ represents the edge weight computed by the adopted weighing function (SW2, SW3, CW2, and CW3). Examples of computing edge weights using these weighting functions were given in Chapter 4. Consequently it can be clearly inferred from the D1 and D2 conditions, that the CMW scheme satisfies the DCP, and that using these two conditions can cut down the search space such that the computation of the mining is lessened considerably.

Procedure subgSpan-CMW(c, $\mathbb{GD}$, σ, θ, $\mathcal{F}$)

```
1  if c ≠ min(c) then
2  |  return
3  end
4  if (sup_GD(c) ≥ σ) ∧ (∀e_i ∈ c, w_GD(e_i) ≥ θ) then
5  |  F ← F ∪ {c}
6  else
7  |  return
8  end
9  C ← ∅
10 Scan GD once, find every edge e such that c can be right-most extended to c ∪ e,
   C ← c ∪ e
11 Sort C in DFS lexicographic order
12 foreach g_k ∈ C do
13 |  if (sup_GD(g_k) ≥ σ) ∧ (∀e_i ∈ g_k, w_GD(e_i) ≥ θ) then
14 |  |  subgSpan-CMW(g_k, GD, σ, θ, F)
15 |  end
16 end
```

5.2.3.1 Pseudo-codes of CMW

In a similar manner to the previous two weighting schemes, the CMW scheme was integrated into the gSpan algorithm (amongst others) to form gSpan-CMW. The pseudo-codes to modify the procedure of 'subgSpan' described in Algorithm 2.3 are presented as a procedure of 'subgSpan-CMW'. In the procedure, a new parameter θ is included to

denote a weighting threshold used in the CMW scheme, and the rest of the parameters reserve the same meaning as those described in Algorithm 2.3. Similar to ATW and AW, two blocks of codes (lines 4-8 and lines 12-16) outline the operation of the CMW scheme.

5.3 Weighting Schemes That Do Not Use Vertex or Edge Weights

The weighting schemes introduced in Section 5.2 all utilized either vertex or edge weights. In this section an alternative weighting scheme, which does not exploit vertex or edge weights, Jaccard Similarity based Weighting (JSW), is proposed. This scheme is investigated in the subsequent sub-section. The advantage offered is that the mechanism obviates the requirement for the adoption of a weighting function.

5.3.1 Jaccard Similarity based Weighting (JSW) scheme

The JSW scheme uses both the standard support measure and a weighting metric which quantifies the significance of each discovered subgraph. The weighting metric adopts the Jaccard similarity measure between two sets A and B. Namely,

$$Jacc(A, B) = \frac{|A \cap B|}{|A \cup B|} \ . \tag{5.8}$$

Definition 5.3.1. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$ and a subgraph g with edges $E(g) = \{e_1, e_2, \cdots, e_k\}$, let two vertexes connected by each $e_i \in E(g)$ equal v_1 and v_2; let the set of graphs where v_1 occurs equal $\Gamma_{\mathbb{GD}}(v_1)$ and the set of graphs where v_2 occurs equal $\Gamma_{\mathbb{GD}}(v_2)$. Then, according to (5.8), the Jaccard similarity coefficient between $\Gamma_{\mathbb{GD}}(v_1)$ and $\Gamma_{\mathbb{GD}}(v_2)$ is defined to quantify the strength of the edge e_i connecting two vertexes v_1 and v_2:*

$$JS(e_i) = Jacc\left(\Gamma_{\mathbb{GD}}(v_1), \Gamma_{\mathbb{GD}}(v_2)\right) = \frac{|\Gamma_{\mathbb{GD}}(v_1) \cap \Gamma_{\mathbb{GD}}(v_2)|}{|\Gamma_{\mathbb{GD}}(v_1) \cup \Gamma_{\mathbb{GD}}(v_2)|} \ . \tag{5.9}$$

Thus, the weight of g, $W_{\mathbb{GD}}(g)$, is accordingly defined as:

$$W_{\mathbb{GD}}(g) = \frac{1}{\sum_{e_i \in E(g)} JS(e_i)} \ . \tag{5.10}$$

When increasing the size of g, the value of the denominator of (5.10) will be increased. Thus, it can be gathered from (5.10) that $W_{\mathbb{GD}}(g)$ is non-increasing with the increase of the size of g (i.e. $W_{\mathbb{GD}}(g)$ holds the DCP). Consequently, $W_{\mathbb{GD}}(g)$ can be further merged into the successive definition.

Definition 5.3.2. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, a support threshold $\sigma \in (0, 1]$, and a weighting threshold $\gamma > 0$, a subgraph g is weighted frequent, if the following two conditions (E1 and E2) are satisfied:*

(E1) $sup_{\mathbb{GD}}(g) \geq \sigma$, *and* **(E2)** $W_{\mathbb{GD}}(g) \geq \gamma$.

Example: Considering the graph data set $\mathbb{GD} = \{G1, G2, G3\}$ shown in Figure 5.3, where the symbol next to each vertex or edge represents its label. Given a subgraph g, which contains two edges $\{a, e\}$, $\Gamma_{\mathbb{GD}}(v1) = \{G1, G3\}$, $\Gamma_{\mathbb{GD}}(v2) = \{G1, G3\}$ and $\Gamma_{\mathbb{GD}}(v6) = \{G1, G2, G3\}$. So, $jC(a) = 1$, $jC(e) = 2/3$ and $W_{\mathbb{GD}}(g) = \frac{1}{1+2/3} \approx 0.6$.

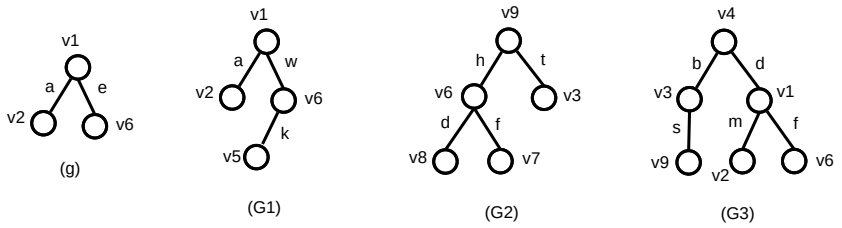


Figure 5.3: An example of computing weights by the JSW scheme

According to Definition 5.3.2, both E1 and E2 maintain the DCP.

Procedure subgSpan-JSW(c, $\mathbb{GD}$, σ, γ, $\mathcal{F}$)

```
1  if c ≠ min(c) then
2  |  return
3  end
4  if sup_GD(c) ≥ σ ∧ W_GD(c) ≥ γ then
5  |  F ← F ∪ {c}
6  else
7  |  return
8  end
9  C ← ∅
10 Scan GD once, find every edge e such that c can be right-most extended to c ∪ e,
   C ← c ∪ e
11 Sort C in DFS lexicographic order
12 foreach g_k ∈ C do
13 |  if sup_GD(g) ≥ σ ∧ W_GD(g) ≥ γ then
14 |  |  subgSpan-JSW(g_k, GD, σ, γ, F)
15 |  end
16 end
```

5.3.1.1 Pseudo-codes of JSW

In a similar manner, the JSW scheme can be incorporated into gSpan to give gSpan-JSW. The pseudo-codes of the modified procedure to describe the incorporation of the JSW scheme are presented in the procedure of 'subgSpan-JSW'. In the procedure, a new parameter γ is introduced to denote a weighting threshold used in the JSW scheme, and the rest of the parameters keep the same meaning as those set in Algorithm 2.3. From lines 4-8 and lines 12-16 in the procedure of 'subgSpan-JSW', if both the support

value and the value of the weighting function for a subgraph candidate are over the respective thresholds, then this subgraph candidate can be further extended to the next level in the search lattice; if either the support value or the value of the weighting function for a subgraph candidate is below the threshold, then this subgraph candidate can be safely discarded. As can be seen from 'subgSpan-JSW', it is relatively simple to adapt the gSpan algorithm to embrace the JSW scheme, compared with weighting schemes that use vertex or edge weights.

5.4 Summary

Four subgraph weighting schemes have been proposed in this chapter: ATW, AW, CMW and JSW. The first three require that either vertex or edge weights are available; the last does not require vertex or edge weights. All these weighting schemes maintain the DCP, so that the search space can be considerably reduced as the frequent subgraph mining progresses, which leads to computational benefits.

Each of these four weighting scheme is applicable under different circumstances using different weighting functions introduced in Chapter 4. More specifically, the ATW scheme requires the SW1, SW4, SW5, and CW1 functions; the AW scheme requires the SW1, SW4, and CW1 functions; the CMW scheme requires the SW2, SW3, CW2, and CW3 functions; the JSW scheme does not need any weighting functions. The incorporation of these four weighting schemes is demonstrated using gSpan as a base algorithm. The pseudo-codes of four weighted gSpan algorithms: gSpan-ATW, gSpan-AW, gSpan-CMW are similar except that the modification of the procedure of 'subgSpan' is different under respective weighting schemes. The procedures: subgSpan-ATW, subgSpan-AW, subgSpan-CMW, and subgSpan-JSW are recursively called by their corresponding weighted gSpan algorithms. Compared with other three weighting schemes, the implementation of JSW is straightforward without recourse to employing user-provided vertex or edge weights or computing weights for vertexes or edges. The experimental analysis and evaluation of these four weighting schemes, using the data sets introduced in Chapter 3, is explored in the following chapter.

Chapter 6

Experimental Study

This chapter reports on the experimental analysis of the four subgraph weighting schemes proposed in the previous Chapter. The objective of this experimental evaluation is to examine: (i) whether or not the weighted FSM algorithms using these four subgraph weighting schemes (i.e. ATW, AW, CMW, and JSW) run faster and identify fewer patterns than the standard FSM algorithms and (ii) whether or not the patterns discovered by the weighted FSM algorithms are as effective as those discovered by the standard FSM algorithms in the context of frequent pattern based classification.

The evaluation of each subgraph weighting scheme consisted of two components. Firstly, the performance of the weighted FSM algorithm, into which the subgraph weighting scheme was incorporated, was compared with the standard FSM algorithms (without any weightings), in terms of the runtime costs and the number of patterns identified. If the runtime costs of the weighted FSM algorithms were similar to or less than those of the standard FSM algorithms, and the number of patterns identified by the former was considerably less than that identified by the latter, this indicated that the weighted FSM algorithms were more efficient than the standard FSM algorithms. Secondly, with respect to data sets that have class labels, the patterns discovered by the weighted and standard FSM algorithms were employed to construct classifiers using the procedure described in Figure 2.9. The performance of the classifiers was compared to see whether the patterns discovered using the weighted FSM algorithms were as effective as those discovered using the standard FSM algorithms. If the results were similar this indicated that the right frequent patterns had been identified.

All the experiments were conducted with a Windows 7 machine using a 2.27GHz Intel Core i5 PC with 3GB main memory, unless otherwise specified. The implementation of all the algorithms used in this chapter is described in Section 6.1. Section 6.2 gives a general description of the data used in this chapter. The evaluation of each subgraph weighting scheme is further studied in Sections 6.3, 6.4, 6.5, and 6.6 respectively.

6.1 Implementation

Three standard FSM algorithms: (i) gSpan, (ii) FFSM, (iii) GASTON were used for the experiments. Among these three, the author re-implemented the gSpan and FFSM algorithms using Java, according to the description of the algorithms found in Yan and Han [2002] and Huan et al. [2003]; the GASTON algorithm was re-implemented by the author using Java, according to the C++ source codes provided by Nijssen and Kok [2004]. With respect to the four subgraph weighting schemes in Chapter 5, the Java implementation of gSpan was used as the base algorithm with which to implement the weighted FSM algorithms: (i) gSpan-ATW, (ii) gSpan-AW, (iii) gSpan-CMW and (iv) gSpan-JSW by integrating four respective weighting schemes into the gSpan algorithm. Specifically, as explained in Section 5.2, gSpan-ATW was coupled with the SW1, SW4, SW5, and CW1 functions; gSpan-AW was coupled with the SW1, SW4, and CW1 functions; gSpan-CMW was coupled with the SW2, SW3, CW2, and CW3 functions; gSpan-JSW did not use any weighting functions. Furthermore, because gSpan was originally intended for undirected graph mining, both gSpan and four weighted FSM algorithms (gSpan-ATW, gSpan-AW, gSpan-CMW, and gSpan-JSW) were modified by the author to accommodate trees and directed graphs in order to test these algorithms on trees and directed graphs.

A variety of classification techniques were used for the evaluation: (i) Decision Trees (i.e. C4.5) [Quinlan, 1993], Naive Bayesian Classifiers (NBCs) [Mitchell, 1996], (iii) Support Vector Machines (SVMs) [Vapnik, 1998], to examine the quality of the discovered patterns. For the NBC and C4.5 classifiers, the WEKA implementations [Hall et al., 2009] were used. For the SVM classifier, the LIBSVM implementation [Chang and Lin, 2001] was used. In the experiments, all the classification results were computed using 10-fold cross validation.

It should also be noted that a maximum memory usage of 3GB was allowed for all the experiments conducted. A symbol of "n/a" is used in the subsequent experimental evaluation to indicate where an FSM algorithm can not complete the mining due to an out-of-memory error.

6.2 Overview of the Data

As described in Chapter 3, ten groups of graph data were used to evaluate the proposed weighted FSM algorithms. Among these, the RT2 and RT3 groups were reserved for the description of two case studies, which will be presented in Chapter 7. Therefore, eight groups, divided into three categories, are considered in this chapter:

1. **Trees**: ST1, ST2, and RT1
2. **Undirected Graphs**: RG1, RG2, and RG3

3. **Directed Graphs**: RG4 and RG5

Since each data group contains more than one data set, a total of 20 data sets were used. These are summarized in Table 6.1. In the table columns 3 to 7 denote, respectively, the number of records, the average number of vertexes, the average number of edges, the number of vertex labels, and the number of edge labels in each data set.

Table 6.1: A summary of data sets employed throughout this chapter

#	Data set	# Transactions	Average $\|V(g)\|$	Average $\|E(g)\|$	$\|L_V\|$	$\|L_E\|$
1	ST1:D10	100000	4	3	100000	1
2	ST1:T1M	1000000	4	3	1000000	1
3	ST2:IM1000-D4	1000	35	34	18	4
4	ST2:IM1000-D5	1000	88	87	18	4
5	ST2:IM1000-D6	1000	193	192	18	4
6	RT1:CSLOGS-ALL	59691	14	13	59691	1
7	RT1:CSLOGS-1	8074	10	9	15652	1
8	RT1:CSLOGS-2	7409	10	9	14413	1
9	RG1:CH1	42682	45	47	63	3
10	RG1:CH2	42247	26	28	67	3
11	RG2:MAM-V80	322	78	765	80	1
12	RG2:MAM-V100	322	96	1230	100	1
13	RG3:BS-V500	500	107	443	499	1
14	RG3:BS-V1000	1000	86	370	991	1
15	RG4:IMDB	7438	34	32	7947	6
16	RG4:Amazon	4305	64	129	6954	6
17	RG4:Ohsumed	3295	55	141	5454	6
18	RG5:Lancashire	105	289	261	2148	10
19	RG5:Scotland	105	933	890	8619	17
20	RG5:GB	105	9957	6754	62626	29

According to Table 6.1, some interesting features of the data employed in this chapter can be observed as follows. The tree data includes ST1, ST2, and RT1. The ST1 (synthetic trees) data includes two, synthetically generated tree data sets: D10 and T1M. The structure of the trees in both data sets is the same, however the size of T1M is 10 times that of D10. The ST2 synthetic image data contains three synthetically generated image data sets: IM1000-D4, IM1000-D5, and IM1000-D6. These three data sets represent the same collection of images but with different levels of decomposition. The trees in IM1000-D6 typically have more vertexes and edges than those in IM1000-D5 which have more vertexes and edges than those in IM1000-D4. The RT1 (Web logs) data describes web usage log files and consists of three data sets: CSLOGS-ALL, CSLOGS-1, and CSLOGS-2. The number of trees in CSLOGS-ALL is much larger than that in CSLOGS-1 or CSLOGS-2. Both CSLOGS-1 and CSLOGS-2 are smaller segments of CSLOGS-ALL. The trees in CSLOGS-ALL also have more vertexes and edges than CSLOGS-1 which has more vertexes and edges than CSLOGS-2.

The undirected graph data includes RG1, RG2, and RG3. The RG1 chemical data contains two data sets: CH1 and CH2. The number of graphs in CH1 is approximately

the same as that in CH2, although the graphs in CH1 have more vertexes and edges than those in CH2, but the latter is denser than the former. The RG2 Mammography data contains two data sets: MAM-V80 and MAM-V100. These two data sets represent the same collection of images with different levels of detail. The graphs in MAM-V100 have more vertexes and edges than those in MAM-V80 but the latter are slightly more dense than the former. The RG3 photographic image data contains two data sets: BS-V500 and BS-V1000. These two data sets represent the same collection of images but with different levels of detail. The graphs in BS-V500 have more vertexes and edges than those in BS-V1000.

The directed graph data comprises the RG4 and RG5 data. The RG4 data contains three text data sets: IMDB, Amazon, and Ohsumed. IMDB is substantially larger than the other two. On average, the graphs in the Amazon data set have more vertexes and edges than Ohsumed, and the graphs in Ohsumed have more vertexes and edges than IMDB. The RG5 social network data comprised three graph collections: Lancashire, Scotland, and GB. Each collection is characterized by edge-weighted and directed graphs. The number of graphs in these three data sets is the same. However, the graphs in the GB data set have significantly more vertexes and edges than the Scotland data set, which in turn has considerably more vertexes and edges than the Lancashire data set.

6.3 The Evaluation of the ATW scheme

The evaluation of ATW was conducted with respect to trees, undirected graphs and directed graphs. Each is discussed in Sub-sections 6.3.1, 6.3.2, and 6.3.3 respectively.

6.3.1 The evaluation of the ATW scheme on trees

For tree data, the gSpan, FFSM, GASTON and gSpan-ATW algorithms were adapted to mine trees only. The data groups that include trees are: ST1, ST2, and RT1. Since the ATW scheme requires vertex or edge weights, for data that did not feature such weights, the vertex and edge weighting functions introduced in Chapter 4 were used to generate appropriate weights. Thus, as explained in Chapter 5.2, the SW1, SW4 and SW5 functions were applied to the ST1, ST2 and RT1 data.

Efficiency test. The performance of the gSpan-ATW algorithm using SW1 on the tree data is shown in Table 6.2. In the table, columns 2 and 7 denote the thresholds used by the standard FSM algorithms and gSpan-ATW respectively, and columns 6 and 9 denote the number of patterns discovered by the respective algorithms (these symbols will be used throughout the rest of this thesis unless otherwise specified). In the table, a range of low support thresholds was used to test gSpan-ATW on ST1 and RT1, because the standard FSM algorithms could only find small sized patterns (i.e

subgraphs with only one vertex). As can be seen in the table, the number of patterns identified by the gSpan-ATW algorithm using SW1 is significantly less than that identified by the standard FSM algorithms but the runtime cost varies with different data sets. More precisely, for the ST1 and RT1 data, when low thresholds (σ and τ) were used, the runtime of gSpan-ATW was considerably faster than that of the standard FSM algorithms; when high thresholds were used, the runtime of gSpan-ATW was close to that of the standard FSM algorithms. For the ST2 data, the runtime of gSpan-ATW is always faster than that of the standard FSM algorithms.

Table 6.2: The performance of gSpan-ATW using SW1 on the ST1, ST2, and RT1 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-ATW + SW1		
		gSpan	FFSM	GASTON		τ(%)	runtime	# patterns
ST1:D10	0.05	10.102	8.657	14.037	21057	0.05	3.931	117
	0.1	7.586	5.927	9.473	7257	0.1	3.863	95
	0.5	4.646	3.465	6.322	727	0.5	3.671	42
	1	3.913	2.743	4.917	196	1	2.964	25
	2	3.341	2.556	4.782	179	2	2.852	22
ST1:T1M	0.05	183.656	167.000	n/a	24492	0.05	104.318	101
	0.1	135.357	115.810	n/a	9493	0.1	110.113	81
	0.5	71.071	51.535	109.163	607	0.5	97.581	41
	1	60.592	36.970	90.239	196	1	95.111	25
	2	59.691	34.053	84.725	178	2	93.633	21
ST2:IM1000-D4	6	142.697	77.36	n/a	435890	6	1.335	390
	8	19.815	10.143	15.752	41398	8	1.162	268
	10	12.146	6.161	9.778	21967	10	0.918	194
	12	5.395	3.067	5.737	7263	12	0.907	165
	14	3.100	1.940	4.638	2782	14	0.891	135
ST2:IM1000-D5	10	563.748	304.526	n/a	489352	10	6.971	978
	12	110.368	67.821	51.791	61333	12	5.404	662
	14	66.628	43.268	33.616	31460	14	4.623	520
	16	42.970	30.006	20.871	16399	16	4.256	433
	18	28.725	22.882	15.286	9215	18	3.734	317
ST2:IM1000-D6	15	592.118	407.344	n/a	112683	15	33.153	2219
	20	308.146	233.924	n/a	49289	20	20.547	1193
	25	132.038	109.170	44.885	14644	25	14.670	769
	30	69.509	67.888	27.517	6936	30	10.593	504
RT1:CSLOGS-ALL	0.3	4.780	3.902	9.920	2268	0.3	3.300	617
	0.4	3.358	2.867	7.319	1134	0.4	3.133	425
	0.5	2.465	2.543	6.650	684	0.5	2.967	304
	0.6	2.311	2.321	5.837	498	0.6	2.959	243
	0.8	2.063	2.062	5.528	311	0.8	2.636	169
RT1:CSLOGS-1	0.2	9.871	3.189	5.117	46104	0.2	0.695	623
	0.4	1.267	0.637	2.050	2582	0.4	0.618	286
	0.6	0.753	0.455	1.789	833	0.6	0.581	175
	0.8	0.623	0.373	1.667	455	0.8	0.566	135
	1	0.554	0.334	1.561	286	1	0.538	106
RT1:CSLOGS-2	0.2	6.388	2.503	3.633	41601	0.2	0.693	662
	0.4	1.080	0.592	1.790	2485	0.4	0.638	292
	0.6	0.679	0.430	1.571	836	0.6	0.588	179
	0.8	0.540	0.350	1.444	462	0.8	0.570	135
	1	0.495	0.299	1.349	280	1	0.557	103

When using SW4 and SW5, the performance of the gSpan-ATW algorithm is shown in Table B.1. In comparison with the standard FSM algorithms, the number of patterns discovered by gSpan-ATW using SW4 or SW5 is considerably less than that discovered

by the standard FSM algorithms. From the table it can be seen that the runtime cost of gSpan-ATW using SW5 on the tree data is higher than that of gSpan-ATW using SW1 or SW4, because the computation of SW5 is more complex than that of SW1 or SW4. This fact is particularly evident for the ST1:T1M data set with 1000000 transactions. Interestingly, in comparison with Table 6.2, Table B.1 indicates that gSpan-ATW using SW5 on ST1:T1M runs slower than gSpan. This suggests that when using the T1M data, the gain obtained by using gSpan-ATW coupled with SW5 is neutralized by the time used to compute the weights using the SW5 weighting function. In addition, for the ST1 data, the number of patterns discovered by gSpan-ATW coupled with SW5 is considerably higher than that discovered by gSpan-ATW coupled with SW4; while for the ST2 data, the number of patterns discovered by the former is smaller than the latter. Further analysis of gSpan-ATW using SW1, SW4, and SW5 on ST2, in comparison with the standard FSM algorithms, is provided in Appendix B.1.1.

Classification evaluation. Given that the trees in the ST2 data have class labels (Seascape vs. Landscape) and that the trees in the RT1:CSLOGS-1(2) data also have class labels (edu vs. other), a two-class classification problem can be identified. The patterns discovered by applying gSpan-ATW to these data were therefore used to construct classifiers (using the process described earlier). The resulting classification accuracy using ST2 and RT1:CSLOGS-1(2), is then presented in 6.3. In the table, the symbols of '#F', 'NBC', 'C4.5', and 'SVM' denote the number of features identified by using the corresponding threshold, the Naive Bayesian Classifier, the decision tree classifier, and the SVM classifier respectively (these symbols will be used throughout the rest of this thesis unless otherwise specified). Because the setting of σ and τ thresholds is different, a different range of thresholds was used to generate effective features for the standard FSM algorithms and gSpan-ATW respectively. As can be seen from the table, the accuracy of the classifiers using patterns discovered by gSpan-ATW with SW1 is very similar to that using patterns discovered by the standard FSM algorithms. However, the number of features used by gSpan-ATW with SW1 is significantly less than that used by the standard FSM algorithms.

Additionally, the accuracy of the classifiers produced using patterns discovered by gSpan-ATW using SW4 and SW5 is further shown in Table B.2. For the ST2 data, using three weighting functions, the classifiers achieved very similar results with a similar number of features. However, compared with the SW1 and SW4 functions, the number of features required to build the classifiers, using gSpan-ATW coupled with SW5, was smaller. As for the RT1:CSLOGS-1(2) data, use of the three weighting functions lead to the same accuracy results with various thresholds.

Table 6.3: The accuracy of the classifiers using patterns discovered by gSpan-ATW using SW1 on the ST2 and RT1:CSLOGS-1(2) data

Dataset	standard FSM algorithms					gSpan-ATW + SW1				
	σ(%)	#F	NBC	SVM	C4.5	τ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	12	7263	92.9	95.4	94.9	2	3020	92.9	95.4	94.9
	14	2782	92.7	95.4	94.6	4	1079	93.0	95.6	94.3
	16	1659	92.6	95.5	94.3	6	390	92.5	95.4	94.2
ST2:IM1000-D5	20	6287	87.0	91.4	91.9	6	3009	86.2	91.4	91.3
	25	2453	86.2	91.4	91.3	8	1501	86.1	91.6	91.3
	30	1163	86.0	91.4	70.8	10	978	85.8	91.4	89.7
ST2:IM1000-D6	30	6936	81.9	76.4	86.8	12	3942	81.2	75.1	86.5
	35	3720	81.2	75.1	86.5	14	2611	81.2	75.1	86.5
	40	1869	81.0	75.2	86.7	16	1881	81.0	75.2	86.7
RT1:CSLOGS-1	0.3	7971	79.8	81.8	81.8	0.2	623	79.8	81.1	80.9
	0.4	2582	79.8	81.8	81.8	0.3	394	79.8	81.1	80.9
	0.5	1286	79.8	81.2	81.2	0.4	286	79.8	80.9	80.9
RT1:CSLOGS-2	0.3	6445	80.4	81.9	82.1	0.2	662	80.4	81.0	81.5
	0.4	2485	80.4	82.0	82.1	0.3	384	80.4	81.7	81.5
	0.5	1281	80.4	81.9	81.9	0.4	292	80.4	81.6	81.5

6.3.2 The evaluation of the ATW scheme on undirected graphs

As noted in Chapter 3, the graphs in the RG1 (Chemical compounds), RG2 (Mammography), and RG3 (Photographic images) data sets are undirected. For the RG1 data, since the graphs in RG1 do not have vertex or edge weights, the SW1, SW4, and SW5 weighting functions were used to generate the required weights. For the RG2 and RG3 data, because the graphs in RG2 and RG3 contained user defined vertex and edge weights, the CW1 weighting function was used instead of any of the structural weighting functions. The detailed experimental analysis of the ATW scheme with respect to the RG1, RG2 and RG3 data is reported below.

Efficiency test. The performance of gSpan-ATW using SW1 on the RG1 data is presented in Table 6.4. From the table it can be seen that both FFSM and GASTON could not operate below a support threshold of 16% while the gSpan algorithm continued. It can also be seen from the table that gSpan-ATW using SW1 requires considerably less runtime to discover far fewer patterns than gSpan using the same range of support thresholds. The performance of gSpan-ATW using SW1 on the CH2 data, as shown in Table 6.4, exhibits a similar advantage over the standard FSM algorithms. Table 6.4 also shows that when applying to RG1:CH2, both gSpan and FFSM can operate below a support threshold of 10% while GASTON fails to operate when the support threshold falls below 10%.

In the cases of SW4 and SW5, Table B.15 indicates that the performance of gSpan-ATW using SW4 or SW5 is very similar to that of gSpan-ATW using SW1. In comparison with SW1, Table B.15 further suggests that gSpan-ATW coupled with SW5 runs faster and discovers fewer patterns than when using SW1 or SW4. An extended analysis of gSpan-ATW using SW1, SW4, and SW5 with a large range of support thresholds,

Table 6.4: The performance of gSpan-ATW using SW1 on the RG1 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-ATW + SW1		
		gSpan	FFSM	GASTON		τ(%)	runtime	# patterns
RG1:CH1	10	1968.265	n/a	n/a	10995	10	272.368	765
	12	1352.122	n/a	n/a	7009	12	241.386	616
	14	989.547	n/a	n/a	4909	14	209.177	516
	16	798.799	n/a	n/a	3596	16	184.728	430
RG1:CH2	4	572.614	740.596	n/a	8624	4	92.606	386
	6	335.605	506.909	n/a	3939	6	73.132	277
	8	243.612	289.341	n/a	2284	8	64.423	211
	10	171.225	242.826	n/a	1502	10	51.010	151

in comparison with the standard FSM algorithms, is presented in Appendix B.2.1.1.

Table 6.5: The performance of gSpan-ATW using CW1-E on the RG2 and RG3 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-ATW + CW1-E		
		gSpan	FFSM	GASTON		τ(%)	runtime	# patterns
RG2:MAM-V80	15	67.208	90.949	83.185	54621	4	10.503	3500
	17	20.110	24.742	22.701	13044	6	8.142	1809
	19	9.381	10.310	8.418	3843	8	6.099	290
	21	8.029	8.425	5.467	2958	10	6.072	83
RG2:MAM-V100	15	167.893	247.678	198.546	95868	4	21.923	5316
	17	53.441	65.258	45.597	22688	6	19.011	3255
	19	21.728	24.707	13.948	5991	8	14.730	646
	21	19.719	21.064	11.282	4784	10	14.725	116
RG3:BS-V500	8	4.992	3.588	8.716	24989	2	1.719	1288
	10	2.278	1.462	3.072	6285	4	1.323	599
	12	1.685	0.779	1.813	2073	6	1.248	495
	14	1.388	0.584	1.482	1229	8	1.128	456
RG3:BS-V1000	6	0.92	0.448	1.332	1404	6	0.668	596
	8	0.671	0.312	1.084	716	8	0.542	479
	10	0.421	0.246	0.958	443	10	0.385	350
	12	0.368	0.133	0.846	238	12	0.259	215

When applying gSpan-ATW to the RG2 data, the CW1 weighting function, which includes both CW1-N for vertex weighting and CW1-E for edge weighting, is employed with respect to the graphs in MAM-V80 and MAM-V100. The performance of gSpan-ATW, coupled with CW1-E, on the RG2 data is shown in Table 6.5. As can be seen in the table, the standard FSM algorithms operate down to a support threshold of 15%, while gSpan-ATW continues down to a support threshold of 4%. Given the fact that the runtime cost of both the standard FSM algorithms and gSpan-ATW increases with the decrease of the support thresholds, Table 6.5 shows that gSpan-ATW with CW1-E runs much faster and identifies significantly lower number of patterns than the standard FSM algorithms, even if the former uses much lower support thresholds than the latter.

Similar to the RG2 data, the graphs in the RG3 have user defined vertex and edge weightings (i.e. CW1-N and CW1-E). Table 6.5 shows the performance of gSpan-ATW coupled with CW1-E on the RG3 data. As can be seen from the table, the number of patterns discovered by gSpan-ATW is significantly less than that discovered by the

standard FSM algorithms but gSpan-ATW runs slightly faster than both gSpan and GASTON, and slightly slower than FFSM.

In addition, the performance of gSpan-ATW coupled with CW1-N on the RG2 and RG3 data is presented in Table B.18. In comparison with Table 6.5, for the RG2 data, gSpan-ATW using CW1-N appears to discover fewer patterns and requires less runtime than when using CW1-E; as for the RG3 data, the results of gSpan-ATW coupled with CW1-N are very close to those obtained using gSpan-ATW coupled with CW1-E.

Classification evaluation. The quality of the discovered patterns by gSpan-ATW on the RG1:CH1, RG2, RG3 data was evaluated, in the same manner as before, using a frequent pattern based classification framework.

For the RG1:CH1 data set, as suggested in Deshpande et al. [2005], a two-class (CA vs. CM) classification problem can be identified (see Section 3.2.4 for a description of the classes). In this classification problem, as suggested by Deshpande et al. [2005], because the size of one class is significantly larger than that of the other class, using the accuracy measure alone to evaluate the classifier is not very accurate. Thus, the area under the ROC (Receiver Operating Characteristic) curve [Provost and Fawcett, 2001], referred to simply as the AUC (Area Under the Curve) score, was also used, in addition to the accuracy measure.

Table 6.6: The accuracy of the classifiers using patterns discovered by gSpan-ATW using SW1 on the CH1 data

Dataset	gSpan					gSpan-ATW + SW1				
	σ(%)	#F	Accuracy			τ(%)	#F	Accuracy		
			NBC	SVM	C4.5			NBC	SVM	C4.5
RG1:CH1	16	3596	77.4	80.1	79.5	4	2277	77.2	80.0	79.8
	18	2735	76.8	79.6	79.2	6	1401	75.3	79.7	79.8
	20	2149	76.8	79.5	80.2	8	1002	73.2	77.9	76.9

The accuracy of the classifiers using the standard FSM algorithms and gSpan-ATW when applied to the CH1 data is given in Tables 6.6 and 6.7. In the tables, different support thresholds were used for the respective standard FSM algorithms and gSpan-ATW, to extract effective patterns for the classification. As can be seen from the tables, the classifiers using patterns discovered by gSpan-ATW achieve very similar performance to that using patterns discovered using the standard FSM algorithms but the former uses considerably fewer features than the latter. Additionally, in comparison with Table 6.6, Table 6.7 suggests that the classifiers built using patterns discovered by gSpan-ATW with SW5 require a smaller number of features than those when using gSpan-ATW with SW1 or SW4, in order to achieve a similar level of accuracy. The AUC "scores" obtained using gSpan-ATW with SW1, SW4 and SW5 when applied to the CH1 data are presented in Tables B.16 and B.17, where very similar behaviour to that shown in Tables 6.6 and 6.7 can be observed.

The graphs in the RG2 data belong to different classes. Thus a two-class (Benign

Table 6.7: The accuracy of the classifiers using patterns discovered by gSpan-ATW using SW4 and SW5 on the CH1 data

Dataset	gSpan-ATW + SW4					gSpan-ATW + SW5				
	τ(%)	#F	Accuracy			τ(%)	#F	Accuracy		
			NBC	SVM	C4.5			NBC	SVM	C4.5
RG1:CH1	4	2382	76.9	79.8	80.1	4	1656	79.1	73.6	78.7
	6	1474	75.4	79.7	79.7	6	1053	77.5	74.4	76.7
	8	1051	73.9	78.6	78.0	8	764	76.1	74.7	77.1

vs. Malignant) classification problem can again be identified to determine whether the patterns discovered by gSpan-ATW using CW1-E are as effective as those discovered by the standard FSM algorithms. The result of applying the generated classifiers to the RG2 data is shown in Table 6.8. In the table, because the settings of the thresholds for the respective standard FSM algorithms and gSpan-ATW using CW1-E are different, lower support thresholds were used for gSpan-ATW in order to extract good quality patterns. As can be seen from the table, the performance of the classifiers using patterns discovered by gSpan-ATW with CW1-E is very similar to that of the classifiers using patterns discovered by the standard FSM algorithms but the former employs considerably fewer features than the latter to build the classifiers, which is an important factor for reducing the computation time of the construction of the classifiers.

Table 6.8: The accuracy of the classifiers using patterns discovered by gSpan-ATW using CW1-E on the RG2 and RG3 data

Dataset	standard FSM algorithms					gSpan-ATW + CW1-E				
	σ(%)	#F	NBC	SVM	C4.5	τ(%)	#F	NBC	SVM	C4.5
RG2:MAM-V80	18	6728	77.4	76.5	71.3	4	3500	74.8	76.5	73.0
	20	3228	74.8	76.5	73.0	5	2719	76.5	75.7	68.7
	22	2698	76.5	75.7	68.7	6	1809	75.7	76.5	73.0
RG2:MAM-V100	18	11097	84.3	80.0	67.8	4	5316	76.5	81.7	71.3
	20	5084	80.0	81.7	72.2	5	4525	80.9	80.9	69.6
	22	4538	80.9	80.9	69.6	6	3255	80.0	81.7	65.2
RG3:BS-V500	8	24989	97.6	95.9	83.5	2	1288	96.5	94.7	83.5
	10	6285	96.5	93.5	84.7	4	599	95.3	92.9	80.0
	12	2073	93.5	92.9	86.5	6	495	95.3	92.9	78.2
RG3:BS-V1000	4	16833	95.9	92.4	84.7	2	911	95.9	92.9	81.2
	6	1404	95.3	91.2	84.1	4	766	94.7	92.4	81.2
	8	716	92.9	91.2	84.1	6	596	93.5	91.8	79.4

The RG3 graphs belong to two classes (Bonsai vs. Sunflower). Thus a two-class problem can again be identified. The patterns discovered by the standard FSM algorithms and gSpan-ATW were used to generate classifiers (three in each case). The accuracy results obtained from using these classifiers are shown in Table 6.8. As can be seen from the table, three classifiers built using patterns discovered using gSpan-ATW with CW1-E achieve very similar accuracy results to those built using patterns discovered using the standard FSM algorithms. However, gSpan-ATW with CW1-E uses far fewer features than the standard FSM algorithms to build the classifiers. For instances,

the largest number of features used by the standard FSM algorithms is 24989, while the largest number of features used by gSpan-ATW with CW1-E is 1288. Considering the performance of the classifiers on both the BS-V500 and BS-V1000 data, the performance of the classifiers for BS-V500 appears to be better than that for BS-V1000. In addition, since both BS-V500 and BS-V1000 represent the same collection of photographic images, it can be inferred that the BS-V500 is a more accurate representation for that collection of images.

When using gSpan-ATW coupled with CW1-N on the RG2 and RG3 data, the performance of the classifiers built using the gSpan-ATW patterns, as shown in Appendix B.2.1.2, is similar to when using gSpan-ATW coupled with CW1-E.

6.3.3 The evaluation of the ATW scheme on directed graphs

As identified in Chapter 3, the graphs in the RG4 (Document base) and RG5 (Social network) data sets are directed. It is relatively straightforward to modify gSpan to accommodate various types of graphs, while FFSM and GASTON do not provide mechanisms to allow their adaptation to the mining of directed graphs. Therefore, only the performance of gSpan and gSpan-ATW could be compared in the context of the evaluation of the ATW scheme with respect to the RG4 and RG5 data.

Efficiency test. When applying gSpan-ATW to the RG4 data, the CW1 weighting function, including CW1-N for vertex weighting and CW1-E for edge weighting, was applicable to graphs in RG4. Because significant patterns could not be found in these data sets when using relatively high support thresholds, low support thresholds were employed.

Table 6.9: The performance of gSpan-ATW using CW1 on the RG4 data

Dataset	gSpan			τ(%)	gSpan-ATW + CW1-E		gSpan-ATW + CW1-N	
	σ(%)	runtime (in seconds)	# patterns		runtime	# patterns	runtime	# patterns
RG4:IMDB	0.1	6.738	8768	0.1	5.125	5523	5.412	5523
	0.2	4.888	3932	0.2	4.413	3319	4.663	3319
	0.4	3.965	1866	0.4	3.698	1780	3.607	1780
	0.6	3.126	1230	0.6	3.358	1203	3.277	1203
RG4:Amazon	0.4	8.184	4680	0.4	7.240	2875	7.617	2875
	0.6	7.341	2901	0.6	6.415	2166	6.845	2166
	0.8	7.051	1974	0.8	6.217	1653	6.764	1653
	1	6.406	1534	1	6.156	1349	6.376	1349
RG4:Ohsumed	0.4	7.391	4138	0.4	6.782	2646	6.429	2646
	0.6	5.762	2630	0.6	5.445	2010	5.548	2010
	0.8	5.366	1902	0.8	4.911	1582	5.243	1582
	1	5.178	1521	1	4.647	1317	5.016	1317

The performance of gSpan-ATW using CW1 on the RG4 data is presented in Table 6.9. In the table, the same range of support thresholds was used for gSpan and gSpan-ATW on each of three data sets that belong to RG4 in order to compare the performance between gSpan and gSpan-ATW. As can be seen in the table, for all three data sets,

gSpan-ATW coupled with CW1-E or CW1-N discovers substantially fewer patterns than gSpan. For the IMDB data, when the support threshold falls below 0.4%, the advantage of gSpan-ATW over gSpan starts to become evident; when the support threshold is over 0.4%, the performance of gSpan-ATW is very similar to that of gSpan. In the case of the Amazon and Ohsumed data sets, the runtime cost of gSpan-ATW is constantly less than that of gSpan. Furthermore, Table 6.9 shows that gSpan-ATW coupled with CW1-E mostly runs slightly faster than gSpan-ATW coupled with CW1-N, but discovers the same number of patterns.

When using the RG5 data, the CW1-E weighting function was used for graphs in each collection. Additionally, the SW5 weighting function was employed to distribute the weights for vertexes of graphs in each collection. Similar to the RG4 data, the same range of support thresholds was used for each data set that belong to RG5. The performance of gSpan-ATW using CW1-E and SW5 on the RG5 data is presented in Table 6.10. It can clearly be seen from the table that gSpan-ATW coupled with CW1-E or SW5 runs faster and identifies significantly fewer patterns than gSpan. In addition, using CW1-E or SW5, there seems to be little difference in the number of patterns identified by gSpan-ATW. However, CW1-E entails the least amount of runtime.

Table 6.10: The performance of gSpan-ATW using CW1-E and SW5 on the RG5 data

Dataset	gSpan			τ(%)	gSpan-ATW + CW1-E		gSpan-ATW + SW5	
	σ(%)	runtime (in seconds)	# patterns		runtime	# patterns	runtime	# patterns
RG5:Lancashire	10	5.691	8107	10	1.103	931	1.324	932
	12	2.891	3520	12	0.994	817	1.249	817
	14	2.010	1898	14	0.923	713	1.177	712
	16	1.575	1211	16	0.887	631	1.164	631
RG5:Scotland	10	43.836	84342	10	2.366	3079	2.983	3079
	12	11.341	20895	12	2.261	2597	2.901	2597
	14	5.531	7856	14	2.152	2235	2.784	2235
	16	3.685	3952	16	2.067	1899	2.591	1899
RG5:GB	15	197.525	59081	15	71.396	16393	80.717	16393
	18	108.26	25248	18	60.525	13432	67.263	13431
	20	79.076	17001	20	52.055	11735	57.971	11735
	22	53.836	11540	22	44.609	9680	52.005	9680

Classification evaluation. The graphs in each of the IMDB, Amazon and Ohsumed data sets are allocated to two classes, a number of two-class classification problems can thus be identified. The patterns discovered by the gSpan and gSpan-ATW algorithms respectively were therefore employed to build classifiers. For the IMDB data, as described in [Ifrim et al., 2008], the classification task was considered to be challenging because the two categories of class labels (Crime vs. Drama) are very close in terms of their theme. The results of the classifiers on the RG4 data using CW1-E are presented in Table 6.11. The performance of the classifiers built using patterns discovered by gSpan-ATW coupled with CW1-N was found to be very similar to that when using gSpan-ATW coupled with CW1-E. Thus, for clarity purpose, only the case of gSpan-

ATW using CW1-E is reported here. The result of the classifiers built using patterns discovered by gSpan-ATW coupled with CW1-N can be found in Appendix B.3.1.1.

Table 6.11: The accuracy of the classifiers using patterns discovered by gSpan-ATW using CW1-E on the RG4 data

Dataset	gSpan					gSpan-ATW + CW1-E				
	σ(%)	#F	NBC	SVM	C4.5	τ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.1	8768	72.9	71.8	73.0	0.2	3319	72.9	72.3	73.1
	0.2	3932	72.9	72.1	73.0	0.4	1780	72.9	72.6	73.1
	0.4	1866	72.9	72.5	73.1	0.6	1203	72.9	72.6	73.1
RG4:Amazon	0.4	4680	92.4	92.7	91.2	0.4	2875	92.4	92.8	91.1
	0.6	2901	92.4	92.7	91.2	0.6	2166	92.4	92.8	91.1
	0.8	1974	92.5	93.0	91.2	0.8	1653	92.5	93.0	91.1
RG4:Ohsumed	0.4	4138	76.9	78.5	74.7	0.4	2646	77.0	79.2	75.1
	0.6	2630	76.9	78.2	74.7	0.6	2010	76.9	79.0	75.1
	0.8	1902	76.9	77.8	74.7	0.8	1582	77.4	79.3	75.5

From Table 6.11 it can be seen that compared with the classifiers generated using patterns discovered by gSpan, the three classifiers built using patterns discovered by gSpan-ATW employ a much smaller number of features to achieve very similar performance.

6.3.4 Summary & discussion

The evaluation of the ATW scheme was conducted by applying gSpan-ATW to different data sets with different weighting functions. Because the implementation of the ATW scheme requires either vertex or edge weightings, for data that did not feature such weightings three structural weighting functions were used. In cases where the data featured content weightings, the content weighting function, CW1, was used instead of the application of any structural weighting functions, because the former was assumed to be more descriptive (accurate in the context of the application) than the latter.

For the RT1 and RG1 data, the graphs were weighted using the SW1, SW4 and SW5 weighting functions. The performance of gSpan-ATW on this data indicates that gSpan-ATW using SW5 is more efficient than when using either SW1 or SW4, in terms of the cost of the runtime and the number of patterns discovered. For the RG2 data, the difference between gSpan-ATW using either vertex or edge weighting is noticeable; while for the RG3, RG4, and RG5 data, the difference between gSpan-ATW using either the vertex weighting or edge weighting is negligible. For other data sets, the performance of gSpan-ATW using different weighting functions is very close.

On the whole, the performance of gSpan-ATW on different types of data indicates that the ATW scheme is effective and efficient with respect to the number of patterns discovered by the standard FSM algorithms. Moreover, the quality of the patterns discovered by gSpan-ATW was at a comparable level with the quality of those discovered using the standard FSM algorithms.

6.4 The Evaluation of the AW scheme

According to Sub-section 5.2.2, the implementation of the AW scheme requires edge weights. Since the data sets included in ST1 (Synthetic trees), ST2 (Synthetic images), RT1 (Web logs), and RG1 (Chemical compounds), do not have edge weights (see Chapter 3), the SW1 and SW4 weighting functions were employed to generate edge weights for these data sets. Additionally, results from experiments (not reported here) indicated that the application of the AW scheme is not suitable with respect to the RT1:CSLOGS-ALL, RG2, RG3:BS-V1000, and RG4 data sets, because gSpan-AW only discovers small sized patterns with respect to these data sets, and furthermore, using different λ thresholds was found to have no effect on restraining the number of patterns discovered, when the support threshold is fixed. Thus, the AW scheme can only be applied to the following data:

- Trees - ST1, ST2, RT1:CSLOGS-1, RT1:CSLOGS-2
- Undirected graphs - RG1, and RG3:BS-V500
- Directed graphs - RG5

The evaluation of the AW scheme on these data sets is examined in the following sub-sections.

6.4.1 The evaluation of the AW scheme on trees

For the experiments on trees, the SW1 and SW4 weighting functions were applied to the tree data: ST1, ST2, RT1:CSLOGS-1 and RT1:CSLOGS-2. Experiments (not reported here) indicated that standard FSM algorithms operated well on the ST1, RT1:CSLOGS-1, and RT1:CSLOGS-2 data, when the support threshold was relatively high. Therefore, low support thresholds were used for the experiments on ST1, RT1:CSLOGS-1, and RT1:CSLOGS-2.

Efficiency test. The performance of gSpan-AW using SW1 on the ST1, ST2 and RT1:CSLOGS-1(2) data is presented in Table 6.12. In the table, columns 2 and 8 denote the support threshold used by the respective standard FSM algorithms and gSpan-AW, and the symbol λ in column 7 denotes the weighting threshold used for the AW scheme. As can be seen in the table the number of patterns discovered by gSpan-AW coupled with SW1 discovers significantly fewer patterns than the standard FSM algorithms, although the runtime cost of the former varies with respect to the different data sets. Generally, for the ST1:D10 and ST2 data, gSpan-AW using SW1 outperforms the standard FSM algorithms with respect to both the runtime cost and the number of patterns discovered; for the ST1:T1M and RT1:CSLOGS-1(2) data, gSpan-AW using SW1 runs faster than three standard FSM algorithms only when using low support

thresholds, because the effort to compute the weightings tends to cancel out any gain achieved by using the AW scheme. In such situations, the gains achieved by gSpan-AW become evident only when the support threshold is relatively low. Table 6.12 also suggests that GASTON fails to operate with low thresholds when applied to ST1:T1M and ST2, due to out-of-memory errors, while gSpan and FFSM continue to operate.

Table 6.12: The performance of gSpan-AW using SW1 on the ST1, ST2, and RT1:CSLOGS-1(2) data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-AW + SW1			
		gSpan	FFSM	GASTON		λ	τ(%)	runtime	# patterns
ST1:D10	0.05	10.102	8.657	14.037	21057	0.01	0.05	3.651	136
	0.1	7.586	5.927	9.473	7257		0.1	3.624	103
	0.5	4.646	3.465	6.322	727		0.5	3.328	41
	1	3.913	2.743	4.917	196		1	2.674	28
	2	3.341	2.556	4.782	179		2	2.503	22
ST1:T1M	0.05	183.656	167.000	n/a	24492	0.01	0.05	68.703	116
	0.1	135.357	115.81	n/a	9493		0.1	66.800	88
	0.5	71.071	51.535	109.163	607		0.5	63.626	39
	1	60.592	36.970	90.239	196		1	61.833	27
	2	59.691	34.053	84.725	178		2	60.466	20
ST2:IM1000-D4	6	142.697	77.36	n/a	435890	0.6	6	1.118	297
	8	19.815	10.143	15.752	41398		8	0.986	217
	10	12.146	6.161	9.778	21967		10	0.867	169
	12	5.395	3.067	5.737	7263		12	0.771	132
	14	3.1	1.940	4.638	2782		14	0.697	105
ST2:IM1000-D5	10	563.748	304.526	n/a	489352	0.6	10	6.018	806
	12	110.368	67.821	51.791	61333		12	5.137	579
	14	66.628	43.268	33.616	31460		14	4.392	472
	16	42.970	30.006	20.871	16399		16	3.947	401
	18	28.725	22.882	15.286	9215		18	3.604	346
ST2:IM1000-D6	15	592.118	407.344	n/a	112683	0.6	15	31.823	2011
	20	308.146	233.924	n/a	49289		20	20.505	1130
	25	132.038	109.170	44.885	14644		25	14.983	724
	30	69.509	67.888	27.517	6936		30	10.297	470
RT1:CSLOGS-1	0.2	9.871	3.189	5.117	46104	0.9	0.2	0.964	1747
	0.4	1.267	0.637	2.050	2582		0.4	0.820	698
	0.6	0.753	0.455	1.789	833		0.6	0.744	439
	0.8	0.623	0.373	1.667	455		0.8	0.736	307
	1	0.554	0.334	1.561	286		1	0.703	252
RT1:CSLOGS-2	0.2	6.388	2.503	3.633	41601	0.9	0.2	0.878	1542
	0.4	1.080	0.592	1.790	2485		0.4	0.757	685
	0.6	0.679	0.43	1.571	836		0.6	0.747	421
	0.8	0.540	0.35	1.444	462		0.8	0.724	316
	1	0.495	0.299	1.349	280		1	0.685	237

When using SW4, the performance of gSpan-AW on the same groups of data is shown in Table B.3. As the table suggests, gSpan-AW using SW4 identifies a considerably smaller number of patterns than when using SW1. However, for the ST1 and ST2:IM1000-D4 data, gSpan-AW coupled with SW4 seems to require more runtime than that when using SW1. Especially for ST1:T1M with 1000000 records, the runtime cost of gSpan-AW using SW4 is lower than both gSpan and FFSM only when the support threshold is below 0.5%, which may indicate that the gain of generating a significantly smaller number of patterns by gSpan-AW using SW4 is at the cost of longer

runtime than the standard FSM algorithms. For the ST2:IM1000-D5, ST2:IM1000-D6, and RT1:CSLOGS-1(2) data, it is evident from the table that gSpan-AW using SW4 is more efficient than that using SW1 in terms of the runtime cost and the number of patterns discovered.

A further analysis of gSpan-AW using SW1 and SW4 on ST2 and RT1:CSLOGS-1(2) with a wide range of support thresholds, in comparison with gSpan, FFSM, and GASTON, is provided in Appendix B.1.2.

Table 6.13: The accuracy of the classifiers using patterns discovered by gSpan-AW with SW1 on the ST2 and RT1:CSLOGS-1(2) data

Dataset	standard FSM algorithms					gSpan-AW + SW1					
	σ(%)	#F	NBC	SVM	C4.5	λ	τ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	12	7263	92.9	95.4	94.9	0.6	2	1388	92.4	95.8	93.9
	14	2782	92.7	95.4	94.6		4	522	92.3	95.3	94.2
	16	1659	92.6	95.5	94.3		6	297	92.5	95.7	94.4
ST2:IM1000-D5	20	6287	87.0	91.4	91.9	0.6	4	4509	88.2	91.9	93.0
	25	2453	86.2	91.4	91.3		6	2000	86.2	91.4	91.3
	30	1163	86.0	91.4	70.8		8	1188	86.1	91.6	91.3
ST2:IM1000-D6	30	6936	81.9	76.4	86.8	0.6	10	5266	84.5	80.2	87.8
	35	3720	81.2	75.1	86.5		15	2011	81.2	75.1	86.5
	40	1869	81.0	75.2	86.7		20	1130	81.0	75.2	86.7
RT1:CSLOGS-1	0.3	7971	79.8	81.8	81.8	0.9	0.2	1747	80.3	81.4	81.7
	0.4	2582	79.8	81.8	81.8		0.4	698	80.3	81.4	81.6
	0.5	1286	79.8	81.2	81.2		0.6	439	80.5	81.3	81.8
RT1:CSLOGS-2	0.3	6445	80.4	81.9	82.1	0.9	0.1	4270	80.6	80.7	82.2
	0.4	2485	80.4	82.0	82.1		0.2	1542	80.6	81.0	82.2
	0.5	1281	80.4	81.9	81.9		0.4	685	80.6	81.9	82.3

Classification evaluation. The classification accuracies obtained by the classifiers generated from patterns obtained using gSpan-AW coupled with SW1, with respect to the ST2 and RT1:CSLOGS-1(2) data are presented in Table 6.13. In the table, different range of support thresholds was used for the respective standard FSM algorithms and gSpan-AW using SW1, in order to extract effective patterns for the classification. Table 6.13 reveals that the performance of the classifiers built using patterns discovered using gSpan-AW is very close to those built using patterns discovered using the standard FSM algorithms. However, the performance of the classifiers built using patterns discovered by gSpan-AW is achieved with a small number of features while the classifiers built using patterns discovered by the standard FSM algorithms require a substantially large number of features.

In comparison with Table 6.13, the performance of the classifiers built using patterns discovered by gSpan-AW coupled with SW4, as shown in Table B.4 further suggests that, when using SW4, the classifiers built using patterns discovered by gSpan-AW require considerably fewer features than when using SW1, but the former achieves slightly lower accuracy than the latter on ST2 and very similar performance to the latter on RT1:CSLOGS-1(2).

6.4.2 The evaluation of the AW scheme on undirected graphs

As noted above, the AW scheme is applicable to only RG1 and RG3:BS-V500. For the RG1 data, because the graphs in RG1 do not have edge weights, the SW1 and SW4 functions were used to generate the required edge weights. For the RG3:BS-V500 data set, the edge weighting, CW1-E, was used to compute the weighting ratio in the AW scheme instead of SW1 or SW4, because the user provided edge weighting (i.e. CW1-E) was assumed to be more representative than edge weights produced using the proposed structural weighting functions (see Chapter 4).

Efficiency test. For the RG1:CH1 data set, although gSpan-AW coupled with SW1 or SW4 discovers a much lower number of patterns and requires significantly less runtime than the standard FSM algorithms, the patterns discovered by the former are less effective than those discovered by the latter, in terms of classification effectiveness. This may suggest that both SW1 and SW4 are not appropriate edge weightings for the graphs in CH1, at least in the context of the AW scheme. There are two possible explanations for this. Firstly, it is generally recognized that using patterns with high frequency do not necessarily lead to good classifications. The weighted support measure (i.e. τ) adopted in the AW scheme tends to miss the patterns with low frequency but high discriminative ability. Therefore, very low support thresholds are required for gSpan-AW, in order to increase the classification accuracy. Secondly, the weighting ratio computed using SW1 or SW4 is not effective for extracting highly discriminative patterns, which play an important role in improving classification accuracy. Accordingly, the experimental result of applying gSpan-AW with SW1 or SW4 on the CH1 data set is omitted in this sub-section (however details are provided in Appendix B.2.2.1).

Tables 6.14 shows the performance of gSpan-AW using SW1 on the CH2 data set. As can be seen from the tables, using the same range of support thresholds, gSpan-AW coupled with SW1 requires far less runtime and discovers a substantially lower number of patterns than gSpan and FFSM while GASTON can not operate below a support threshold of 10%, due to the out-of-memory error.

Table 6.14: The performance of gSpan-AW using SW1 on the CH2 data

Dataset	σ(%)	runtime (in seconds)			# patterns	λ	gSpan-AW + SW1		
		gSpan	FFSM	GASTON			τ(%)	runtime	# patterns
RG1:CH2	4	572.614	740.596	n/a	8624	0.4	4	47.379	112
	6	335.605	506.909	n/a	3939		6	38.461	80
	8	243.612	289.341	n/a	2284		8	34.904	67
	10	171.225	242.826	n/a	1502		10	31.160	59

Additionally, Table B.23 indicates that gSpan-AW coupled with SW4 on CH2 appears to be slightly more efficient than that coupled with SW1, in terms of the runtime cost and the number of patterns discovered. A direct comparison between gSpan-AW using SW1 or SW4 and the standard FSM algorithms on CH2 is further shown in

Figure B.21.

The performance of gSpan-AW using CW1-E on the RG3:BS-V500 data is shown in Table 6.15. In the table, the performance results using the standard FSM algorithms are not available because they can not operate below a support threshold of 8%. However, it can be seen from the table that the runtime and the number of patterns discovered by gSpan-AW decreased with the growth of the support threshold (when the λ threshold is fixed).

Table 6.15: The performance of of gSpan-AW with CW1-E on the RG3:BS-V500 data

Dataset	σ(%)	runtime (in seconds)			# patterns	λ	gSpan-AW + CW1-E		
		gSpan	FFSM	GASTON			τ(%)	runtime	# patterns
RG3:BS-V500	1	n/a	n/a	n/a	n/a	0.4	1	2.986	4923
	2	n/a	n/a	n/a	n/a		2	1.807	1171
	4	n/a	n/a	n/a	n/a		4	1.572	588
	6	n/a	n/a	n/a	n/a		6	1.445	493

Classification evaluation. The patterns discovered using gSpan-AW with CW1-E when applied to the BS-V500 data were used to construct frequent pattern based classifiers. The accuracy results obtained using these classifiers are listed in Table 6.16. As can be seen from the table, the three classifiers built using patterns discovered by gSpan-AW achieve fairly good results but require a significantly smaller number of features than when using patterns discovered by the standard FSM algorithms to build the classifiers.

Table 6.16: The accuracy of the classifiers using patterns discovered by gSpan-AW using CW1-E on the RG3:BS-V500 data

Dataset	standard FSM algorithms					λ	gSpan-AW + CW1-E				
	σ(%)	#F	NBC	SVM	C4.5		τ(%)	#F	NBC	SVM	C4.5
RG3:BS-V500	8	24989	97.6	95.9	83.5	0.4	1	4923	96.5	95.9	82.4
	10	6285	96.5	93.5	84.7		2	1171	96.5	93.5	83.5
	12	2073	93.5	92.9	86.5		4	588	95.3	92.9	80.0

6.4.3 The evaluation of the AW scheme on directed graphs

As noted previously the AW scheme is not applicable to the RG4 data. Thus, only the performance of gSpan and gSpan-AW on RG5 was compared. Because the graphs in RG5 had user defined edge weightings, CW1-E was used to compute the weighting ratio in the AW scheme.

Efficiency test. The performance of gSpan-AW on the RG5 data is presented in Table 6.17. From the table it can be clearly seen that the performance of gSpan-AW with CW1-E is better than gSpan in terms of the runtime cost and the number of patterns identified. For the largest data set, GB, when the support threshold is low (i.e. below 22%) the advantage of gSpan-AW over gSpan becomes prominent with respect

to the runtime cost and the number of patterns identified; when the support threshold is at 22%, the runtime difference between gSpan-AW and gSpan becomes negligible.

The RG5 data did not feature class labels therefore classification accuracy tests could not be conducted.

Table 6.17: The performance of gSpan-AW using CW1-E on the RG5 data

Dataset	gSpan			gSpan-AW + CW1-E			
	σ(%)	runtime (in seconds)	# patterns	λ	τ(%)	runtime	# patterns
RG5:Lancashire	10	5.691	8107	0.6	10	1.562	1082
	12	2.891	3520		12	1.396	923
	14	2.010	1898		14	1.208	765
	16	1.575	1211		16	1.079	669
RG5:Scotland	10	43.836	84342	0.6	10	5.762	3847
	12	11.341	20895		12	2.977	2767
	14	5.531	7856		14	2.546	2287
	16	3.685	3952		16	2.287	1906
RG5:GB	15	197.525	59081	0.6	15	136.352	17485
	18	108.26	25248		18	83.668	13831
	20	79.076	17001		20	67.422	11979
	22	53.836	11540		22	52.863	9770

6.4.4 Summary & discussion

The evaluation of the AW scheme was conducted by testing the performance of gSpan-AW in comparison with the performance of three standard FSM algorithms. Because the implementation of the AW scheme required edge weights, effective (accurate) edge weighting functions are the key to the effectiveness of the AW scheme.

In the evaluation of the AW scheme, the SW1 and SW4 weighting functions were used to compute edge weights for data which did not feature the required edge weights. For some data sets, either the SW1 or SW4 function worked well; but for other data sets, these two weighting functions did not work effectively. The reason for this is that for some data sets, such as the ST1:D10, ST2, RT1:CSLOGS-1, RT1:CSLOGS-2, and RG1:CH2, the SW1 or SW4 function can accurately quantify the relationship between two vertexes connecting the edge, resulting in an effective weighting ratio $r(g)$ used in the AW scheme. For some other data sets such as ST1:T1M, the advantage of gSpan-AW over gSpan is evident only when using low support thresholds, because the gain achieved using the AW scheme is neutralized by the effort to compute SW1 or SW4 in a very large data set. For some other data sets, such as RG1:CH1, although gSpan-AW is efficient, both the weighted support and the weighting ratio computed using either SW1 or SW4 were found to be ineffective for extracting patterns with discriminative power, which lead to ineffective classifications. Thus the goodness of the weighting function used in the AW scheme is actually dependent on the features of the graph data.

For graphs which have a user defined edge weighting such as RG3:BS-V500 and RG5, the evaluation of gSpan-AW indicated that using the AW scheme can substantially

reduce the large number of patterns that would otherwise be identified by the standard FSM algorithms.

Overall we can conclude that the effectiveness of the AW scheme requires a good edge weighting function, and that how to choose the appropriate edge weighting function is data dependent.

6.5 The Evaluation of the CMW scheme

The CMW scheme was coupled with four edge weighting functions: SW2, SW3, CW2, and CW3, which were introduced in Chapter 4. Among these four weighting functions, SW2 and SW3 were applicable to data without predefined class labels, and CW2 and CW3 to data with predefined class labels. Consequently, the data sets used for the evaluation of the CMW scheme can be categorized into two "divisions" according to whether the data sets contain class labels or not.

- **Division A:** data sets that do not have class labels
  - Trees - ST1 (Synthetic trees) and RT1:CSLOGS-ALL (Web logs)
  - Undirected graphs - RG1:CH2 (Chemical compounds)
  - Directed graphs - RG5 (Social network)
- **Division B:** data sets that have class labels
  - Trees - ST2 (Synthetic images), RT1:CSLOGS-1(2) (Web logs)
  - Undirected graphs - RG1:CH1 (Chemical compounds), RG2 (Mammography), and RG3 (Photographic images)
  - Directed graphs - RG4 (Document base)

Hence, for data sets belong to Division A, SW2 and SW3 were employed to compute edge weights; for data sets belong to Division B, CW2 and CW3 were used. SW2 and SW3 can also be applied to Division B; however CW2 and CW3 were clearly not applicable to data sets belong to Division A (because data sets in Division A do not have class labels). In the following reported experiments CW2 and CW3 were used in relation to Division B, except in certain cases when using SW2 and SW3 was found to be advantageous.

The evaluation of the CMW scheme on the data sets that belong to these two divisions is reported in the following sub-sections, in terms of trees, undirected graphs, and directed graphs respectively.

6.5.1 The evaluation of the CMW scheme on trees

The tree data used for the experiments using the CMW scheme included ST1, ST2, RT1. According to the features of the trees, the performance analysis of the gSpan-CMW algorithm on trees belong to Division A is reported first, followed by the performance analysis of gSpan-CMW on trees belong to Division B.

Efficiency test. The performance of gSpan-CMW on the ST1 and RT1:CSLOGS-ALL data is presented in Table 6.18. In the table, the symbol θ denotes the weighting threshold used for the CMW scheme. It can be clearly seen from the table that gSpan-CMW using SW2 on ST1 discovers substantially fewer patterns than the standard FSM algorithms. For the ST1:D10 data set, gSpan-CMW using SW2 runs constantly faster than the standard FSM algorithms; for the ST1:T1M data, gSpan-CMW using SW2 consistently runs faster than gSpan and GASTON, but runs slower than FFSM when using support thresholds of over 0.5%. It should also be noted that both standard FSM algorithms and gSpan-CMW used 3GB of memory to carry out the mining on the ST1:T1M data in order to obtain an appropriate comparison.

Table 6.18: The performance of gSpan-CMW using SW2 on the ST1 and RT1:CSLOGS-ALL data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-CMW + SW2			
		gSpan	FFSM	GASTON		θ	σ(%)	runtime	# patterns
ST1:D10	0.05	10.102	8.657	14.037	21057	0.6	0.05	4.534	132
	0.1	7.586	5.927	9.473	7257		0.1	4.251	112
	0.5	4.649	3.465	6.322	727		0.5	2.594	56
	1	3.913	2.743	4.917	196		1	2.530	31
	2	3.341	2.556	4.782	179		2	2.474	30
ST1:T1M	0.05	183.656	167.000	n/a	24492	0.6	0.05	68.771	107
	0.1	135.357	115.810	n/a	9493		0.1	63.551	89
	0.5	71.071	51.535	109.163	607		0.5	53.105	39
	1	60.592	36.970	90.239	196		1	47.482	31
	2	59.691	34.053	84.725	178		2	46.716	29
RT1:CSLOGS-ALL	0.3	4.780	3.902	9.920	2268	0.6	0.3	2.759	732
	0.4	3.358	2.867	7.319	1134		0.4	2.420	514
	0.5	2.465	2.543	6.650	684		0.5	2.148	361
	0.6	2.311	2.321	5.837	498		0.6	1.770	288
	0.8	2.063	2.062	5.528	311		0.8	1.734	200

Since the size of the RT1:CSLOGS-ALL data set is very large, and both gSpan-CMW and the standard FSM algorithms can operate well on the CSLOGS-ALL data, a range of low support thresholds was used to evaluate the performance of gSpan-CMW on CSLOGS-ALL. It can be seen from Table 6.18 that gSpan-CMW using SW2 performs better than the three standard FSM algorithms in terms of the runtime cost and the number of patterns discovered.

When gSpan-CMW using SW3 was applied to the same groups of data, a very similar performance can be observed in Table B.7. However, in comparison with Table 6.18, gSpan-CMW using SW3 appears to run slightly faster and identifies a smaller number of patterns than that using SW2.

Table 6.19: The performance of gSpan-CMW using CW2 on the ST2 and RT1:CSLOGS-1(2) data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-CMW + CW2			
		gSpan	FFSM	GASTON		θ	σ(%)	runtime	# patterns
ST2:IM1000-D4	6	142.697	77.36	n/a	435890	3	6	0.780	181
	8	19.815	10.143	15.752	41398		8	0.686	142
	10	12.146	6.161	9.778	21967		10	0.639	109
	12	5.395	3.067	5.737	7263		12	0.562	82
	14	3.100	1.940	4.638	2782		14	0.530	72
ST2:IM1000-D5	10	563.748	304.526	n/a	489352	3	10	10.047	2022
	12	110.368	67.821	51.791	61333		12	8.144	1682
	14	66.628	43.268	33.616	31460		14	6.052	1260
	16	42.970	30.006	20.871	16399		16	3.978	680
	18	28.725	22.882	15.286	9215		18	2.933	410
ST2:IM1000-D6	15	592.118	407.344	n/a	112683	3	15	21.263	2203
	20	308.146	233.924	n/a	49289		20	17.691	1725
	25	132.038	109.170	44.885	14644		25	13.650	1237
	30	69.509	67.888	27.517	6936		30	7.067	539
RT1:CSLOGS-1	0.2	9.871	3.189	5.117	46104	50	0.2	1.965	695
	0.4	1.267	0.637	2.050	2582		0.4	1.045	316
	0.6	0.753	0.455	1.789	833		0.6	0.905	194
	0.8	0.623	0.373	1.667	455		0.8	0.749	141
	1	0.554	0.334	1.561	286		1	0.640	109
RT1:CSLOGS-2	0.2	6.388	2.503	3.633	41601	50	0.2	1.740	735
	0.4	1.080	0.592	1.790	2485		0.4	0.966	319
	0.6	0.679	0.430	1.571	836		0.6	0.760	196
	0.8	0.540	0.350	1.444	462		0.8	0.721	141
	1	0.495	0.299	1.349	280		1	0.655	107

For the ST2 data, the performance of gSpan-CMW using CW2 is shown in Table 6.19. As can be seen in the table, gSpan-CMW using CW2 clearly discovers significantly fewer patterns with considerably less runtime than the standard FSM algorithms.

The trees in the RT1:CSLOGS-1 and RT1:CSLOGS-2 data both have a similar structure. The standard FSM algorithms, gSpan, FFSM, and GASTON operated well on CSLOGS-1 and CSLOGS-2 except when the support threshold was very low. Table 6.19 also shows the performance of gSpan-CMW using CW2 on the RT1:CSLOGS-1 and RT1:CSLOGS-2 data sets. It can be observed from the table that gSpan-CMW coupled with CW2 runs slower than gSpan and FFSM when the support threshold is over 0.4% and that the former runs faster than the latter when the support threshold is below 0.4%. The reason for this behaviour is that the gain achieved by gSpan-CMW using relatively high support thresholds is less than the time needed to compute the weightings using CW2.

The performance of gSpan-CMW coupled with CW3 when applied to the ST2 data is shown in Table B.8. For the ST2:IM1000-D4 data set, Table B.8 shows that the performance of gSpan-CMW using CW3 is very close to that of gSpan-CMW using CW2. However, for the ST2:IM1000-D5 and ST2:IM1000-D6 data sets, Table B.8 shows that the number of patterns discovered by gSpan-CMW using CW3 is significantly smaller than that discovered by gSpan-CMW using CW2, and the former runs faster than the latter. In the case of using the CSLOGS-1 and CSLOGS-2 data sets, the

performance of gSpan-CMW using CW3, as shown in Table B.9, is similar to that of gSpan-CMW using CW2 but gSpan-CMW coupled with CW3 discovers fewer patterns than gSpan-CMW coupled with CW2.

Table 6.20: The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the ST2 and RT1:CSLOGS-1(2) data

Dataset	standard FSM algorithms					gSpan-CMW + CW2					
	σ(%)	#F	NBC	SVM	C4.5	θ	σ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	12	7263	92.9	95.4	94.9	3	2	731	89.6	95.0	95.0
	14	2782	92.7	95.4	94.6		4	360	89.6	94.9	95.2
	16	1659	92.6	95.5	94.3		6	181	89.6	94.9	95.2
ST2:IM1000-D5	20	6287	87.0	91.4	91.9	5	2	1514	87.8	89.2	90.7
	25	2453	86.2	91.4	91.3		4	834	87.2	89.1	90.5
	30	1163	86.0	91.4	70.8		6	559	86.3	88.9	88.7
ST2:IM1000-D6	30	6936	81.9	76.4	86.8	3	10	4901	83.2	79.0	87.5
	35	3720	81.2	75.1	86.5		15	2203	81.9	78.5	86.9
	40	1869	81.0	75.2	86.7		20	1725	82.3	77.5	88.9
RT1:CSLOGS-1	0.3	7971	79.8	81.8	81.8	50	0.1	1376	80.7	81.1	82.1
	0.4	2582	79.8	81.8	81.8		0.2	695	80.7	81.6	82.2
	0.5	1286	79.8	81.2	81.2		0.3	437	80.7	81.8	82.2
RT1:CSLOGS-2	0.3	6445	80.4	81.9	82.1	50	0.1	1425	80.4	81.1	82.6
	0.4	2485	80.4	82.0	82.1		0.2	735	80.4	81.5	82.6
	0.5	1281	80.4	81.9	81.9		0.3	426	80.4	81.4	82.5

Classification evaluation. The patterns discovered by gSpan-CMW using CW2 when applied to the ST2 and RT1:CSLOGS-1(2) data were used to construct the classifier in the same manner as before. The accuracy results obtained using these classifiers are shown in Table 6.20. In the table, the symbol θ (column 7) denotes the weighting threshold used with respect to the CMW scheme. Different support thresholds were used for the respective standard FSM algorithms and gSpan-CMW with CW2 in order to extract optimal patterns for the classification. Referring to the performance of the classifiers built using patterns discovered by the standard FSM algorithms, Table 6.20 indicates that the classifiers using patterns discovered by gSpan-CMW coupled with CW2 achieve very similar performance but use a substantially smaller number of features.

In the case of using CW3, Table B.10 shows the accuracy results of the classifiers constructed using patterns discovered by gSpan-CMW on the same groups of data. As can be inferred from Tables 6.20 and B.10, using the CW3 function appears to be more efficient than CW2 with respect to building the classifiers on ST2 and RT1:CSLOGS-1(2), because the classifiers built using patterns identified by gSpan-CMW with CW3 require much fewer features than those built using patterns identified by gSpan-CMW with CW2.

6.5.2 The evaluation of the CMW scheme on undirected graphs

For undirected graphs: RG1, RG2, and RG3, the graphs in RG1:CH2 belong to Division A while the graphs in RG1:CH1, RG2, and RG3 belong to Division B. Thus, the performance of gSpan-CMW using SW2 or SW3 on RG1:CH2 is described firstly, and then followed by the performance of gSpan-CMW using CW2 or CW3 on RG1:CH1, RG2, and RG3.

Efficiency test. As noted previously, both gSpan and FFSM work well on the CH2 data set. The performance of gSpan-CMW using SW2 or SW3 is presented in Appendix B.2.3. However, it is interesting to note in Tables B.24 and B.25 that gSpan-CMW, when coupled with either SW2 or SW3, runs slower than both gSpan and FFSM, although the former identifies substantially fewer patterns than the latter. The reason for this phenomenon may be that the time used to compute edge weights using SW2 or SW3 is more than the time saved by reducing the search space using the CMW scheme. Thus, there is a trade-off between the number of patterns identified and the cost of the mining. Is it worthwhile to discover a considerably smaller number of patterns but at a greater runtime cost? The answer to this question is application dependent. For instances, it can be seen from Table B.24 that at a support threshold of 4%, gSpan requires about 10 minutes to find more than 8000 patterns while gSpan-CMW uses about 20 minutes to find approximately 2000 patterns. Considering the difference in runtime between the two algorithms the advantage of the reduced number of patterns identified by gSpan-CMW over gSpan is pronounced. Additionally, using SW2 or SW3, gSpan-CMW appears to identify the same number of patterns with support thresholds between 4% and 10%.

Table 6.21 presents the performance of gSpan-CMW with CW2 in comparison with the standard FSM algorithms on the CH1, RG2, and RG3 data. In the table, the same range of support thresholds was used for the corresponding standard FSM algorithms and gSpan-CMW with CW2, in order to easily compare their performance. As can be seen in the table, for all the data sets, gSpan-CMW using CW2 discovers significantly fewer patterns than the standard FSM algorithms.

For the CH1 data set, as indicated by Table 6.21, both FFSM and GASTON were unable to operate below a support threshold of 16%, due to out-of-memory errors while gSpan-CMW using CW2 ran much faster than gSpan.

Recall that the RG2 Mammography data comprises MAM-V80 and MAM-V100. Standard FSM algorithms operate well on the RG2 data. Under the support thresholds between 15% and 21%, gSpan-CMW using CW2 appears to run considerably faster than both gSpan and FFSM. In the case of MAM-V100, gSpan-CMW using CW2 runs slightly slower than GASTON when using the support thresholds of over 17%. The reason for this fact is that the advantage of GASTON is more prominent than the gain achieved by gSpan-CMW with CW2 when using relatively high support thresholds.

Table 6.21: The performance of gSpan-CMW with CW2 on the RG1:CH1, RG2, and RG3 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-CMW + CW2			
		gSpan	FFSM	GASTON		θ	σ(%)	runtime	# patterns
RG1:CH1	10	1968.265	n/a	n/a	10995	8	10	12.059	36
	12	1352.122	n/a	n/a	7009		12	9.632	27
	14	989.547	n/a	n/a	4909		14	9.527	27
	16	798.799	n/a	n/a	3596		16	9.188	25
RG2:MAM-V80	15	67.208	90.949	83.185	54621	2	15	10.192	2777
	17	20.110	24.742	22.701	13044		17	6.958	1078
	19	9.381	10.310	8.418	3843		19	6.783	537
	21	8.029	8.425	5.467	2958		21	6.277	477
RG2:MAM-V100	15	167.893	247.678	198.546	95868	2	15	25.341	6962
	17	53.441	65.258	45.597	22688		17	17.810	2446
	19	21.728	24.707	13.948	5991		19	15.583	928
	21	19.719	21.064	11.282	4784		21	15.515	819
RG3:BS-V500	8	4.992	3.588	8.716	24989	2	8	4.905	11307
	10	2.278	1.462	3.072	6285		10	2.478	3565
	12	1.685	0.779	1.813	2073		12	1.676	1397
	14	1.388	0.584	1.482	1229		14	1.447	909
RG3:BS-V1000	6	0.920	0.448	1.332	1404	4	6	1.014	1063
	8	0.671	0.312	1.084	716		8	0.792	619
	10	0.421	0.246	0.958	443		10	0.573	400
	12	0.368	0.133	0.846	238		12	0.464	227

As noted previously the RG3 photographic data contains two data sets: BS-V500 and BS-V1000. The standard FSM algorithms operated well on both data sets. Table 6.21 indicates that the runtime cost of gSpan-CMW using CW2 is very close to that of gSpan and slightly slower than FFSM while GASTON is the slowest. The reason for this is that when using relatively high support thresholds, the benefit of the CMW scheme is not prominent since the gain obtained by gSpan-CMW is cancelled out by the effort to compute the edge weights using CW2.

Table B.26 further shows that the performance of gSpan-CMW coupled with CW3 is very similar to that of gSpan-CMW using CW2. However, for RG2 and RG3, gSpan-CMW coupled with CW3 seems to discover considerably fewer patterns than that coupled with CW2.

Classification evaluation. The classification results using gSpan-CMW with CW2 on the CH1 data are presented in Tables 6.22. Compared with the performance of the classifiers built using patterns discovered by the standard FSM algorithms, Table 6.22 demonstrates that the classifiers built using patterns discovered by gSpan-CMW achieve a similar level of performance to those built using patterns discovered by the standard FSM algorithms but with a significantly smaller number of features. Using an alternative measure, AUC, the performance of the classifiers built using patterns discovered by gSpan-CMW with CW2 is provided in Table B.27.

Although the performance of the classifiers obtained using patterns discovered by gSpan-CMW with CW3 is very close to that when using CW2, it can be inferred from Table B.28 that the classifiers built using patterns discovered by gSpan-CMW with

CW3 require a considerably smaller number of features than those built using patterns discovered by gSpan-CMW with CW2.

Table 6.22: The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the CH1 data

Dataset	gSpan					θ	gSpan-CMW + CW2				
	σ(%)	#F	Accuracy				σ(%)	#F	Accuracy		
			NBC	SVM	C4.5				NBC	SVM	C4.5
RG1:CH1	16	3596	77.4	80.1	79.5	8	0.1	1516	76.8	78.3	81.0
	18	2735	76.8	79.6	79.2		0.2	830	76.9	78.5	81.4
	20	2149	76.8	79.5	80.2		0.4	491	76.9	78.8	81.4

The classification results obtained using patterns generated by gSpan-CMW with CW2 from RG2 are listed in Tables 6.23. Compared with the performance of the classifiers built using patterns discovered by the standard FSM algorithms, Table 6.23 shows that the classifiers built using the gSpan-CMW patterns undoubtedly achieve higher accuracy results using significantly fewer features. In comparison with gSpan-CMW coupled with CW2, Table B.29 suggests that the classifiers built using patterns discovered by gSpan-CMW with CW3 employ a far lower number of features, than those when using gSpan-CMW with CW2, to attain a very similar classification performance.

Table 6.23: The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the RG2 and RG3 data

Dataset	standard FSM algorithms					θ	gSpan-CMW + CW2				
	σ(%)	#F	NBC	SVM	C4.5		σ(%)	#F	NBC	SVM	C4.5
RG2:MAM-V80	18	6728	77.4	76.5	71.3	2	15	2777	85.2	73.9	73.0
	20	3228	74.8	76.5	73.0		17	1078	78.3	75.7	61.7
	22	2698	76.5	75.7	68.7		19	537	73.0	73.9	63.5
RG2:MAM-V100	18	11097	84.3	80.0	67.8	2	15	6962	94.8	84.3	72.2
	20	5084	80.0	81.7	72.2		17	2446	80.0	77.4	62.6
	22	4538	80.9	80.9	69.6		19	928	73.0	78.3	72.2
RG3:BS-V500	8	24989	97.6	95.9	83.5	4	10	2674	96.5	95.3	82.9
	10	6285	96.5	93.5	84.7		12	1153	97.6	94.1	82.4
	12	2073	93.5	92.9	86.5		14	776	95.3	92.9	82.9
RG3:BS-V1000	4	16833	95.9	92.4	84.7	4	6	1063	94.7	90.6	82.9
	6	1404	95.3	91.2	84.1		8	619	94.1	90.0	84.7
	8	716	92.9	91.2	84.1		10	400	92.9	91.8	85.3

When using gSpan-CMW with CW2 on RG3, the classification accuracy results as shown in Table 6.23, indicate that the classifiers built using patterns discovered by gSpan-CMW with CW2 perform similarly to those built using patterns discovered by the standard FSM algorithms, but use a substantially smaller number of features, which is usually an big advantage in terms of the computational effort to construct the classifiers. Table B.29 shows the similar performance of the classifiers when using gSpan-CMW with CW3.

6.5.3 The evaluation of the CMW scheme on directed graphs

Recall that the directed graphs used for the evaluation of the CMW scheme are the RG4 document base data and the RG5 social network data. Due to the difference of the mechanisms adopted using the three standard FSM algorithms, only gSpan could be modified to handle directed graphs. Thus, both gSpan and gSpan-CMW were adapted to mine directed graphs and only the performance of these two algorithms were reported in this sub-section.

Efficiency test. Since the graphs in RG4 have class labels, the CW2 and CW3 functions were used to compute edge weights for the graphs. However, gSpan-CMW coupled with either CW2 or CW3 required more runtime, in order to identify fewer patterns than gSpan. More interestingly, the patterns discovered by gSpan-CMW coupled with either CW2 or CW3 were found to be as effective as those discovered by gSpan in terms of classification accuracy (see Appendix B.3.2.1 for details). Thus, is it rewarding to spend more time discovering a considerable smaller number of patterns while maintaining the quality of the patterns? As suggested in Sub-section 6.5.2, the answer to this is very much application dependent.

Table 6.24: The performance of gSpan-CMW using SW2 and SW3 on the RG4 and RG5 data

Dataset	gSpan			σ(%)	gSpan-CMW + SW2			gSpan-CMW + SW3		
	σ(%)	runtime (in seconds)	# patterns		θ	runtime	# patterns	θ	runtime	# patterns
RG4:IMDB	0.1	6.738	8768	0.1	0.8	5.452	5536	0.2	5.909	5676
	0.2	4.888	3932	0.2		4.547	3323		4.863	3338
	0.4	3.965	1866	0.4		3.505	1780		3.513	1780
	0.6	3.126	1230	0.6		3.270	1203		3.297	1203
RG4:Amazon	0.4	8.184	4680	0.4	0.4	7.613	2891	0.1	7.871	2914
	0.6	7.341	2901	0.6		6.797	2177		7.021	2195
	0.8	7.051	1974	0.8		6.631	1662		6.805	1673
	1	6.406	1534	1		6.143	1354		6.317	1362
RG4:Ohsumed	0.4	7.391	4138	0.4	0.6	7.088	2674	0.2	7.358	2693
	0.6	5.762	2630	0.6		5.415	2026		5.372	2035
	0.8	5.366	1902	0.8		4.989	1592		5.042	1597
	1	5.178	1521	1		4.800	1327		4.827	1332
RG5:Lancashire	10	5.691	8107	10	0.4	1.373	949	0.1	1.272	960
	12	2.891	3520	12		1.317	832		1.152	841
	14	2.010	1898	14		1.261	725		1.094	732
	16	1.575	1211	16		1.231	639		1.029	643
RG5:Scotland	10	43.836	84342	10	0.8	3.467	3094	0.6	2.753	3094
	12	11.341	20895	12		3.291	2608		2.526	2608
	14	5.531	7856	14		3.078	2242		2.410	2242
	16	3.685	3952	16		2.966	1905		2.204	1905
RG5:GB	15	197.525	59081	15	0.8	82.956	16487	0.8	75.281	16450
	18	108.26	25248	18		69.348	13497		62.016	13471
	20	79.076	17001	20		61.067	11785		56.766	11766
	22	53.836	11540	22		46.309	9716		48.332	9705

If the class labels for graphs are ignored, the SW2 and SW3 functions can also be applied to the RG4 data. The performance of gSpan-CMW coupled with SW2 or SW3 is shown in Tables 6.24. For the IMDB data set shown in Table 6.24, when the support threshold is at 0.6%, gSpan-CMW runs slightly slower than gSpan; when the support

threshold is below 0.4%, gSpan-CMW starts to run faster than gSpan. In addition, gSpan-CMW identifies significantly fewer patterns than gSpan only when the support threshold is below 0.4%. The reason for this behaviour is that the gain obtained from using the CMW scheme is cancelled out by the effort to compute the weightings using the CMW scheme when using relatively high support thresholds to discover small sized patterns. Table 6.24 also indicates that gSpan-CMW behaves similarly on the Amazon and Ohsumed data sets (i.e. gSpan-CMW outperforms gSpan in terms of both runtime and the number of patterns identified). Furthermore, it can be inferred from Table 6.24 that gSpan-CMW coupled with SW2 mostly requires less runtime than when coupled with SW3 but discovers a similar number of patterns.

When gSpan-CMW was applied to the RG5 data, the performance of gSpan-CMW, as shown in Table 6.24, suggests that gSpan-CMW coupled with SW2 or SW3 spends less runtime and identifies significantly fewer patterns than gSpan with the same range of support thresholds. Further, gSpan-CMW coupled with SW2 discovers a very similar number of patterns to that when using SW3. Further analysis of the performance of gSpan-CMW on RG5 using a wide range of support thresholds, in comparison with gSpan, is provided in Appendix B.3.2.2.

Table 6.25: The accuracy of the classifiers using patterns discovered by gSpan-CMW with SW2 on the RG4 data

Dataset	gSpan					gSpan-CMW + SW2					
	σ(%)	#F	NBC	SVM	C4.5	θ	σ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.1	8768	72.9	71.8	73.0	0.8	0.2	3323	72.9	72.3	73.1
	0.2	3932	72.9	72.1	73.0		0.4	1780	72.9	72.6	73.1
	0.4	1866	72.9	72.5	73.1		0.6	1203	72.9	72.6	73.1
RG4:Amazon	0.4	4680	92.4	92.7	91.2	0.4	0.4	2891	92.4	92.7	91.2
	0.6	2901	92.4	92.7	91.2		0.6	2177	92.4	92.7	91.2
	0.8	1974	92.5	93.0	91.2		0.8	1662	92.5	93.0	91.2
RG4:Ohsumed	0.4	4138	76.9	78.5	74.7	0.6	0.4	2674	76.9	78.7	74.7
	0.6	2630	76.9	78.2	74.7		0.6	2026	76.9	78.4	74.7
	0.8	1902	76.9	77.8	74.7		0.8	1592	76.9	77.8	74.7

Classification evaluation. The patterns discovered by gSpan-CMW using SW2 on RG4 were again used to construct frequent pattern based classifiers. The accuracy results obtained is shown in Table 6.25. As can be seen from the table, the performance of the classifiers using patterns discovered by gSpan-CMW is very similar to that using patterns discovered by gSpan but the former requires a much smaller number of features to achieve the same level of accuracy. A very similar performance was obtained using the classifiers built using patterns discovered by gSpan-CMW using SW3 as can be observed from Table B.36.

6.5.4 Summary & discussion

The evaluation of the CMW scheme was conducted using gSpan-CMW with four different weighting functions, SW2, SW3, CW2, and CW3, on a number of different data sets. The data sets used in the evaluation were divided into two groups: (i) Division A comprising data sets that do not have associated class labels and (ii) Division B comprising data sets that do have associated class labels. Accordingly, the SW2 and SW3 weighting functions were applied to data sets in Division A, and the CW2 and CW3 weighting functions to the data sets in division B.

For the SW2 and SW3 weighting functions, the computation is relatively simpler than that for the CW2 and CW3 weighting functions. However, on the RG1:CH2 data set, although gSpan-CMW identifies considerably fewer patterns than the standard FSM algorithms, the former requires more runtime than the latter. This fact may suggest that the benefit of using the CMW scheme is cancelled out by the effort to compute the edge weights through the use of the SW2 or SW3 functions. In the case of the other data sets that do not have class labels, e.g. ST1, RT1:CSLOGS-ALL, and RG5, gSpan-CMW coupled with SW2 or SW3 still maintains their advantage over gSpan, with respect to the runtime and the number of patterns discovered. Additionally, coupling with the SW3 function seems to be more efficient than when using SW2.

For the CW2 and CW3 weighting functions, the computation required knowledge of the class labels. When applied to data such as ST2, RG1:CH1, and RG2, gSpan-CMW coupled with either CW2 or CW3 identifies fewer patterns and requires less runtime than gSpan. More importantly, the patterns discovered by gSpan-CMW on ST2 and RG2 are as effective as those discovered by gSpan in terms of the classification result. As for the RG1:CH1 data set, the classifiers built using patterns discovered by gSpan-CMW with CW2 or CW3 achieved lower accuracy than those built using patterns discovered by gSpan, although the accuracy of the former is reasonable good (e.g. 76%-81%).

For the RT1:CSLOGS-1, RT1:CSLOGS-2, and RG3 data, gSpan-CMW coupled with CW2 or CW3 runs faster than gSpan only when using low support thresholds, and when using relatively high support thresholds, the former runs slightly slower than the latter. However, the patterns discovered by gSpan-CMW on these data sets still maintain the same quality as those discovered by the standard FSM algorithms. As mentioned in the previous sub-sections, the reason for the behaviour of gSpan-CMW on RT1:CSLOGS-1(2) and RG3 is that the advantage of gSpan using high support thresholds is more prominent than the gain achieved by gSpan-CMW on these data. In addition, coupling with the CW3 function appears to be more efficient than when coupling with CW2.

On the RG4 data, coupled with either CW2 or CW3, gSpan-CMW failed to demon-

strate any advantage over gSpan, because the computation cost using CW2 or CW3 exceeded the gain achieved using the CMW scheme. Thus gSpan-CMW coupled with SW2 or SW3 was used in the evaluation. The performance of gSpan-CMW coupled with SW2 or SW3 indicates that gSpan-CMW outperforms gSpan, with respect to both the runtime cost and the number of patterns discovered. Thus indicating that when the CW2 or CW3 weighting function is not effective with respect to the CMW scheme, the SW2 or SW3 function can be used as an alternative.

6.6 The Evaluation of the JSW scheme

As described in Section 5.3.1, the JSW scheme does not require any vertex or edge weightings. Thus none of the weighting functions introduced in Chapter 4 are applicable. In a similar manner as described above the JSW evaluation comprised the following:

- The evaluation of the JSW scheme on trees.
- The evaluation of the JSW scheme on undirected graphs.
- The evaluation of the JSW scheme on directed graphs.

The details of the evaluations, with respect to each of the above, are presented in Sub-sections 6.6.1, 6.6.2, and 6.6.3 respectively.

6.6.1 The evaluation of the JSW scheme on trees

Three groups of tree data, ST1, ST2, and RT1, were used in the evaluation. The experimental results obtained with respect to each group of data are presented below.

Efficiency test. The performance of gSpan-JSW on the ST1:D10 is shown in Table 6.26. In the table, the symbol σ (columns 2 and 8) denotes the support threshold used for the standard FSM algorithms and gSpan-JSW, and the symbol γ denotes the weighting threshold used with respect to the JSW scheme. From the table it can be seen that gSpan-JSW using $\gamma = 8$ on D10 runs faster and discovers considerably fewer patterns than the three standard FSM algorithms.

For the T1M data, both standard FSM algorithms and gSpan-JSW used 3GB of memory in order to compare the runtime cost appropriately. It can be seen in Table 6.26 that gSpan-JSW using $\gamma = 10$ runs faster and discovers significantly fewer patterns than the standard FSM algorithms while GASTON fails to proceed at the support threshold of below 0.1%. The performance of gSpan-JSW using a smaller γ value on the ST1 data, as shown in Table B.11, indicates that gSpan-JSW with a smaller γ value needs more runtime than both gSpan and FFSM, and is less efficient than gSpan-JSW with a larger γ value.

Table 6.26 further shows the performance of gSpan-JSW on the CSLOGS-ALL data. From the table it can be seen that: (i) the GASTON algorithm runs the slowest; (ii) using the same range of support thresholds (0.3% to 0.8%), gSpan-JSW discovers far fewer patterns with less runtime than the three standard FSM algorithms. Referring to Figure B.11, gSpan-JSW with a larger γ value appears to run faster and discover fewer patterns than that with a smaller γ value.

Table 6.26: The performance of gSpan-JSW on the ST1 and RT1:CSLOGS-ALL data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-JSW			
		gSpan	FFSM	GASTON		γ	σ(%)	runtime	# patterns
ST1:D10	0.05	10.102	8.657	14.037	21057	8	0.05	7.988	165
	0.1	7.586	5.927	9.473	7257		0.1	5.289	107
	0.5	4.649	3.465	6.322	727		0.5	2.917	32
	1	3.913	2.743	4.917	196		1	2.496	22
	2	3.341	2.556	4.782	179		2	2.481	20
ST1:T1M	0.05	183.656	167.000	n/a	24492	10	0.05	165.719	134
	0.1	135.357	115.810	n/a	9493		0.1	115.102	94
	0.5	71.071	51.535	109.163	607		0.5	59.331	31
	1	60.592	36.970	90.239	196		1	56.616	22
	2	59.691	34.053	84.725	178		2	54.629	19
RT1:CSLOGS-ALL	0.3	4.780	3.902	9.920	2268	10	0.3	3.328	804
	0.4	3.358	2.867	7.319	1134		0.4	2.781	554
	0.5	2.465	2.543	6.650	684		0.5	2.396	393
	0.6	2.311	2.321	5.837	498		0.6	2.213	309
	0.8	2.063	2.062	5.528	311		0.8	1.900	206

The performance of gSpan-JSW on the ST2 data is shown in Table 6.27. As can be seen from the table, the GASTON algorithm fails to operate using relatively low support thresholds, because of high demands on memory usage; the performance of gSpan-JSW is very similar across the data sets; gSpan-JSW identifies significantly fewer patterns and requires far less runtime than the standard FSM algorithms. A further analysis of gSpan-JSW on ST2, using a wide range of thresholds, is also provided in Appendix B.1.4.2.

The performance of gSpan-JSW on CSLOGS-1 and CSLOGS-2 is presented in Table 6.27. From the table it can be noted that: (i) the FFSM algorithm runs faster than the other algorithms; (ii) the runtime of gSpan-JSW is very similar to that of gSpan and FFSM; and (iii) gSpan-JSW starts to run faster than FFSM when the support threshold drops to below 0.4%. The reason for this is that the gain obtained by using the JSW scheme is not very distinct when the computation cost of applying gSpan to certain data sets is minimal. In other words, the benefit of using the JSW scheme is neutralized by the effort to compute the weighting, when the application of algorithms such as gSpan entails very little computational overhead. Table 6.27 also indicates that gSpan-JSW identifies substantially fewer patterns than standard FSM algorithms. Further, in comparison with the performance of gSpan-JSW using a smaller γ value as shown in Table B.12, it can be inferred that gSpan-JSW with a larger γ value is more efficient than that when using a smaller γ value.

Table 6.27: The performance of gSpan-JSW on the ST2 and RT1:CSLOGS-1(2) data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-JSW			
		gSpan	FFSM	GASTON		γ	σ (%)	runtime	# patterns
ST2:IM1000-D4	6	142.697	77.36	n/a	435890	0.6	6	0.987	213
	8	19.815	10.143	15.752	41398		8	0.871	166
	10	12.146	6.161	9.778	21967		10	0.784	153
	12	5.395	3.067	5.737	7263		12	0.776	139
	14	3.100	1.940	4.638	2782		14	0.703	115
ST2:IM1000-D5	10	563.748	304.526	n/a	489352	0.6	10	1.676	284
	12	110.368	67.821	51.791	61333		12	1.476	243
	14	66.628	43.268	33.616	31460		14	1.242	192
	16	42.970	30.006	20.871	16399		16	1.109	149
	18	28.725	22.882	15.286	9215		18	1.104	131
ST2:IM1000-D6	15	592.118	407.344	n/a	112683	0.6	15	3.277	356
	20	308.146	233.924	n/a	49289		20	2.550	208
	25	132.038	109.170	44.885	14644		25	2.157	135
	30	69.509	67.888	27.517	6936		30	1.797	102
RT1:CSLOGS-1	0.2	9.871	3.189	5.117	46104	10	0.2	1.239	815
	0.4	1.267	0.637	2.050	2582		0.4	0.748	361
	0.6	0.753	0.455	1.789	833		0.6	0.572	216
	0.8	0.623	0.373	1.667	455		0.8	0.505	164
	1	0.554	0.334	1.561	286		1	0.480	127
RT1:CSLOGS-2	0.2	6.388	2.503	3.633	41601	10	0.2	1.142	870
	0.4	1.080	0.592	1.790	2485		0.4	0.677	372
	0.6	0.679	0.430	1.571	836		0.6	0.549	221
	0.8	0.540	0.350	1.444	462		0.8	0.508	165
	1	0.495	0.299	1.349	280		1	0.485	125

Table 6.28: The accuracy of the classifiers using patterns discovered by gSpan-JSW on the ST2 and RT1:CSLOGS-1(2) data

Dataset	standard FSM algorithms					gSpan-JSW					
	σ(%)	#F	NBC	SVM	C4.5	γ	σ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	12	7263	92.9	95.4	94.9	0.4	2	2001	90.6	95.7	95.5
	14	2782	92.7	95.4	94.6		4	1176	90.6	95.7	95.4
	16	1659	92.6	95.5	94.3		6	823	90.6	95.7	95.4
ST2:IM1000-D5	20	6287	87.0	91.4	91.9	0.4	4	2155	84.1	90.2	89.8
	25	2453	86.2	91.4	91.3		6	1528	83.6	90.0	90.4
	30	1163	86.0	91.4	70.8		8	1077	82.9	89.9	89.9
ST2:IM1000-D6	30	6936	81.9	76.4	86.8	0.4	5	3526	81.3	78.4	88.8
	35	3720	81.2	75.1	86.5		10	1634	81.0	75.7	89.6
	40	1869	81.0	75.2	86.7		15	838	80.1	74.5	89.0
RT1:CSLOGS-1	0.3	7971	79.8	81.8	81.8	5	0.4	417	79.8	81.8	81.8
	0.4	2582	79.8	81.8	81.8		0.6	242	79.8	81.2	81.8
	0.5	1286	79.8	81.2	81.2		0.8	177	80.7	80.9	81.3
RT1:CSLOGS-2	0.3	6445	80.4	81.9	82.1	5	0.4	429	80.5	81.8	82.3
	0.4	2485	80.4	82.0	82.1		0.6	247	80.5	81.4	82.0
	0.5	1281	80.4	81.9	81.9		0.8	186	80.6	81.6	81.9

Classification evaluation. The patterns discovered by gSpan-JSW on the ST2 and RT1:CSLOGS-1(2) data were used to build frequent pattern based classifiers so as to verify the quality of the discovered patterns. The classification accuracies obtained, with respect to the ST2 data are presented in Table 6.28. In the table, a different range of support thresholds was used for the respective standard FSM algorithms and gSpan-JSW in order to extract appropriate features for the classification. As can be seen

from the table, the classifiers built using patterns discovered by gSpan-JSW achieved a similar accuracy to those built using patterns discovered by the standard FSM algorithms, but the former employed a substantially smaller number of features to construct the classifiers than the latter. This fact suggests that gSpan-JSW can discover fewer patterns without compromising the quality of the patterns.

Table 6.28 further lists the classification results obtained, with respect to the CSLOGS-1 and CSLOGS-2 data sets. From the table it can be seen that classifiers obtained using patterns detected by gSpan-JSW achieve slightly higher accuracy results, with far fewer features, than those obtained using patterns discovered by the standard FSM algorithms.

6.6.2 The evaluation of the JSW scheme on undirected graphs

Three groups of data, RG1, RG2 and RG3, that feature undirected graphs were used to evaluate the operation of the JSW scheme with respect to undirected graphs. The experimental results with respect to each data set are presented in detail in the following paragraphs.

Efficiency test. The performance of the gSpan-JSW algorithm on the RG1:CH1 data is shown in Table 6.29. In the table, both FFSM and GASTON failed to complete when using support thresholds of below 16%, while gSpan was able to proceed using a support threshold of 10%. However, gSpan-JSW, when coupled with $\gamma = 0.6$, runs significantly faster and identifies considerably fewer patterns than gSpan. When gSpan-JSW is applied to the RG1:CH2 data, the performance of gSpan-JSW, as shown in Table 6.29, indicated that both gSpan and FFSM ran significantly slower, and discovered substantially more patterns than gSpan-JSW using $\gamma = 0.6$, while GASTON was unable to proceed at support thresholds of below 16%. Further analysis of gSpan-JSW using different γ values and the RG1 data can also be found in Appendix B.2.4.1.

The RG2 Mammography data contains two data sets: MAM-V80 and MAM-V100. These two data sets are two different representations, with different degrees of precision, for the same collection of data. The three standard FSM algorithms operated well when applied to the MAM-V80 and MAM-V100 data sets. Table 6.29 presents the performance of gSpan-JSW using these two data sets. As can be seen from the table, gSpan-JSW starts to run faster than the three standard FSM algorithms when the support threshold is below 19%, while the former runs slightly slower than the latter when the support threshold is over 19%. The reason for this is that the advantage of the JSW scheme is not discernible when gSpan can actually work well using relatively high support thresholds, i.e. the gain obtained by the JSW scheme is cancelled out by the effort to compute the weightings when using relatively high support thresholds. Table 6.29 also shows that using gSpan-JSW on RG2 results in a significant decrease in the number of discovered patterns when the support threshold is below 19%.

Table 6.29: The performance of gSpan-JSW on the RG1, RG2, and RG3 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-JSW			
		gSpan	FFSM	GASTON		γ	σ(%)	runtime	# patterns
RG1:CH1	10	1968.265	n/a	n/a	10995	0.6	10	16.396	34
	12	1352.122	n/a	n/a	7009		12	14.355	29
	14	989.547	n/a	n/a	4909		14	13.919	25
	16	798.799	n/a	n/a	3596		16	13.193	24
RG1:CH2	4	572.614	740.596	n/a	8624	0.6	4	18.957	58
	6	335.605	506.909	n/a	3939		6	14.868	42
	8	243.612	289.341	n/a	2284		8	11.933	32
	10	171.225	242.826	n/a	1502		10	10.509	26
RG2:MAM-V80	15	67.208	90.949	83.185	54621	0.6	15	12.495	3529
	17	20.110	24.742	22.701	13044		17	9.419	3239
	19	9.381	10.310	8.418	3843		19	8.447	3192
	21	8.029	8.425	5.467	2958		21	8.168	2943
RG2:MAM-V100	15	167.893	247.678	198.546	95868	0.6	15	34.210	5762
	17	53.441	65.258	45.597	22688		17	24.557	5114
	19	21.728	24.707	13.948	5991		19	22.004	4977
	21	19.719	21.064	11.282	4784		21	21.322	4761
RG3:BS-V500	8	4.992	3.588	8.716	24989	0.8	8	3.164	10858
	10	2.278	1.462	3.072	6285		10	1.912	4418
	12	1.685	0.779	1.813	2073		12	1.368	1858
	14	1.388	0.584	1.482	1229		14	1.149	1160
RG3:BS-V1000	4	2.660	1.618	5.486	16833	0.8	4	2.224	10262
	6	0.920	0.448	1.332	1404		6	0.905	1403
	8	0.671	0.312	1.084	716		8	0.640	715
	10	0.421	0.246	0.958	443		10	0.421	443

The RG3 photographic data contains two data sets: BS-V500 and BS-V1000. These two data sets represent the same collection of images but with different levels of decomposition. The three standard FSM algorithms operate very well on both BS-V500 and BS-V1000. The performance of gSpan-JSW on these two data sets is presented in Table 6.29. From the table, it can be seen that gSpan-JSW runs faster than both gSpan and GASTON although the difference in runtime is small, and the former discovers far fewer patterns than the standard FSM algorithms. In the case of BS-V1000, the JSW scheme is not effective when using relatively high support thresholds, because both standard FSM algorithms and gSpan-JSW tend to discover a very similar number of small sized patterns. Further, the advantage of gSpan-JSW over gSpan on BS-V1000 is noticeable only when using low support thresholds (e.g. below 6%).

Table 6.30: The accuracy of the classifiers using patterns discovered by gSpan-JSW on the CH1 data

Dataset	gSpan					gSpan-JSW					
	σ(%)	#F	Accuracy			γ	σ(%)	#F	Accuracy		
			NBC	SVM	C4.5				NBC	SVM	C4.5
RG1:CH1	16	3596	77.4	80.1	79.5	0.6	0.1	783	77.1	77.3	79.0
	18	2735	76.8	79.6	79.2		0.2	404	77.1	77.6	79.0
	20	2149	76.8	79.5	80.2		0.4	248	77.4	77.4	77.0

Classification evaluation. The classification accuracy result obtained using gSpan-JSW on CH1, in comparison with that using gSpan, is presented in Table 6.30. Simi-

larly, the AUC measure, which represents the area under the ROC curve, is also used to quantify the performance of the classifier. As can be seen from the table, the classifiers using patterns discovered by gSpan-JSW achieve a similar level of performance to those built using patterns discovered by gSpan but need a significantly smaller number of features. The AUC measure of the classifiers, as shown in Table B.30, indicates that the AUC scores obtained using gSpan-JSW are slightly lower than those obtained using gSpan.

In the case of RG2 and RG3, the patterns discovered by gSpan-JSW were used to construct frequent pattern based classifiers. The accuracy results obtained are presented in Table 6.31. As can be seen from the table, the classifiers built using pattens discovered by gSpan-JSW achieved a similar level of accuracy to those built using patterns discovered by the standard FSM algorithms, but the former use a substantially lower number of features.

Table 6.31: The accuracy of the classifiers using patterns discovered by gSpan-JSW on the RG2 and RG3 data

Dataset	standard FSM algorithms					γ	gSpan-JSW				
	σ(%)	#F	NBC	SVM	C4.5		σ(%)	#F	NBC	SVM	C4.5
RG2:MAM-V80	18	6728	77.4	76.5	71.3	0.6	21	2943	76.5	75.7	68.7
	20	3228	74.8	76.5	73.0		23	2226	75.7	76.5	73.0
	22	2698	76.5	75.7	68.7		25	1276	77.4	71.3	71.3
RG2:MAM-V100	18	11097	84.3	80.0	67.8	0.6	21	4761	80.9	80.9	69.6
	20	5084	80.0	81.7	72.2		23	3903	80.0	80.0	70.4
	22	4538	80.9	80.9	69.6		25	2495	80.0	81.7	65.2
RG3:BS-V500	8	24989	97.6	95.9	83.5	0.6	8	10858	98.2	95.3	88.2
	10	6285	96.5	93.5	84.7		10	4418	97.1	95.3	85.3
	12	2073	93.5	92.9	86.5		12	1858	97.1	95.3	85.3
RG3:BS-V1000	4	16833	95.9	92.4	84.7	0.8	4	10262	96.5	91.8	82.9
	6	1404	95.3	91.2	84.1		6	1403	95.3	91.2	82.9
	8	716	92.9	91.2	84.1		8	715	94.1	90.0	84.7

6.6.3 The evaluation of the JSW scheme on directed graphs

To evaluate the performance of the JSW scheme when applied to directed graphs the RG4 (Document base) and RG5 (Social network) data were used. For reasons already discussed, only the comparisons between gSpan and gSpan-JSW are reported in this section. The experimental results obtained using RG4 and RG5 are described in the following.

Efficiency test. From the foregoing the RG4 data contains three data sets: IMDB, Amazon, and Ohsumed. gSpan operates very well on these three data sets. Thus only low support thresholds were used in order to show the benefits of using the JSW scheme. The performance of gSpan-JSW on the RG4 is shown in Table 6.32. From the table, gSpan-JSW discovers far fewer patterns than gSpan for IMDB, Amazon, and Ohsumed. For the IMDB data set, when the support threshold drops to below 0.6%, gSpan-JSW starts to display a better runtime performance; when the support threshold is at 0.6%,

gSpan runs slightly faster than gSpan-JSW and the two algorithms discover a very similar number of patterns. The reason for this behaviour is that the time saved by the JSW scheme is usually less than the time needed to compute the weightings when using relative high support thresholds. As for the Amazon and Ohsumed data set, gSpan-JSW required less runtime than gSpan.

Table 6.32: The performance of gSpan-JSW on the RG4 and RG5 data

Dataset	gSpan			gSpan-JSW			
	σ(%)	runtime (in seconds)	# patterns	γ	σ(%)	runtime	# patterns
RG4:IMDB	0.1	6.738	8768	30	0.1	6.466	5828
	0.2	4.888	3932		0.2	4.560	3333
	0.4	3.965	1866		0.4	3.530	1780
	0.6	3.126	1230		0.6	3.280	1203
RG4:Amazon	0.4	8.184	4680	25	0.4	8.101	2915
	0.6	7.341	2901		0.6	6.779	2170
	0.8	7.051	1974		0.8	6.306	1653
	1	6.406	1534		1	6.109	1349
RG4:Ohsumed	0.4	7.391	4138	40	0.4	7.274	2671
	0.6	5.762	2630		0.6	5.293	2015
	0.8	5.366	1902		0.8	4.955	1583
	1	5.178	1521		1	4.747	1317
RG5:Lancashire	10	5.691	8107	0.8	10	4.574	2163
	12	2.891	3520		12	2.278	1383
	14	2.010	1898		14	1.625	1004
	16	1.575	1211		16	1.248	787
RG5:Scotland	10	43.836	84342	0.8	10	3.313	3771
	12	11.341	20895		12	2.860	2963
	14	5.531	7856		14	2.598	2463
	16	3.685	3952		16	2.407	2060
RG5:GB	15	197.525	59081	0.8	15	80.269	18079
	18	108.260	25248		18	64.749	14275
	20	79.076	17001		20	58.513	12308
	22	53.836	11540		22	48.494	10003

Table 6.32 further shows the performance of gSpan-JSW when applied to the RG5 data. As can be seen from the table, gSpan-JSW runs faster and identifies significantly fewer patterns than gSpan on the Lancashire, Scotland, and GB data sets. Appendix B.3.3.1 provides a more extended comparison with gSpan of the operation of gSpan-JSW using different γ values.

Classification evaluation. The classification results using patterns discovered by gSpan-JSW when applied to the RG4 data, in comparison with those using patterns discovered by gSpan, are presented in Table 6.33. It can be seen from the table that the performance of classifiers built using patterns discovered by gSpan-JSW is slightly higher than that of classifiers built using patterns discovered by gSpan on the IMDB and Ohsumed data, and is the same as that of classifiers built using patterns discovered by gSpan on the Amazon data. However, the number of features employed to build the classifiers using gSpan-JSW is considerably less than when using gSpan.

Table 6.33: The accuracy of the classifiers using patterns discovered by gSpan-JSW on the RG4 data

Dataset	gSpan					gSpan-JSW					
	σ(%)	#F	NBC	SVM	C4.5	γ	σ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.1	8768	72.9	71.8	73.0	30	0.1	5828	72.9	71.9	73.1
	0.2	3932	72.9	72.1	73.0		0.2	3333	72.9	72.3	73.1
	0.4	1866	72.9	72.5	73.1		0.4	1780	72.9	72.6	73.1
RG4:Amazon	0.4	4680	92.4	92.7	91.2	25	0.4	2915	92.4	92.7	91.2
	0.6	2901	92.4	92.7	91.2		0.6	2170	92.4	92.7	91.2
	0.8	1974	92.5	93.0	91.2		0.8	1653	92.5	93.0	91.2
RG4:Ohsumed	0.4	4138	76.9	78.5	74.7	40	0.4	2671	77.4	79.3	75.4
	0.6	2630	76.9	78.2	74.7		0.6	2015	77.4	79.0	75.4
	0.8	1902	76.9	77.8	74.7		0.8	1583	77.4	78.6	75.4

6.6.4 Summary & discussion

The evaluation of the JSW scheme was conducted using gSpan-JSW to test its performance with respect to different data sets. The application of the JSW scheme requires two thresholds: (i) support, and (ii) γ. If the γ value is set to zero, gSpan-JSW degrades to the classic gSpan algorithm. Therefore, the γ threshold determines how efficient gSpan-JSW is. Based on the experiments shown in the above sub-sections, the most desirable γ value varies with the different data sets. In the reported experiments the most appropriate γ value with respect to each data set was chosen using a 'generate-and-test' process.

Compared with the performance of standard FSM algorithms (e.g. gSpan or FFSM), gSpan-JSW generally identified fewer patterns in a shorter runtime. The patterns discovered by gSpan-JSW were found to be as effective as those discovered by the standard FSM algorithms when they were used to build classifiers. In some cases, the performance of the classifiers built using patterns discovered by gSpan-JSW was even better than that obtained using patterns discovered by gSpan.

Considering the results of using gSpan-JSW on real data sets, more details are given as follows. For the RT1, RG2, RG3, and RG4 data, the size of these data sets varied from hundreds to tens of thousands of graphs/trees. The standard FSM algorithm, gSpan, worked very well on these data sets. In this case, the issue of the high computation cost associated with the use of standard FSM algorithms becomes negligible. Thus, the small gain achieved by the JSW scheme is usually neutralized by the effort to compute the weightings, to the extent that gSpan was often more efficient than gSpan-JSW, especially when using high support thresholds. However, a range of classifiers built using patterns discovered by gSpan-JSW still achieved a very similar performance to those built using patterns discovered by the standard FSM algorithms. This fact indicates that the advantage of gSpan-JSW over the standard FSM algorithms is relatively small for data sets where the standard FSM algorithms can operate well.

For the RG1 and RG5:GB data, the graphs in these data sets have a substantially

large number of vertexes and edges. The standard FSM algorithms can be applied to the mining task; however, they frequently require a considerable amount of runtime to complete. Using the JSW scheme, gSpan-JSW can discover fewer patterns with substantially less runtime.

On the whole, the evaluation of the JSW scheme described in Sub-sections 6.6.1, 6.6.2, and 6.6.3 demonstrated that gSpan-JSW when coupled with appropriate γ values is more efficient than the standard FSM algorithms, and that the patterns discovered by gSpan-JSW are mostly as effective as those discovered by the standard FSM algorithms.

6.7 Summary

The empirical results obtained from the evaluation of the four subgraph weighting schemes introduced in Chapter 5 were explored in this chapter. Since each subgraph weighting scheme was applicable to different types of data sets, the evaluation of each of the four subgraph weighting schemes was carried out, according to the characteristics of the data sets described in Chapter 3, in conjunction with the weighting functions advanced in Chapter 4. One common feature shared by these four subgraph weighting schemes is that the DCP is maintained, which can be utilized to reduce the computation, resulting in more efficient mining. However, it is usually difficult to devise a subgraph weighting scheme that maintains the DCP. Therefore, an alternative subgraph weighting scheme, which does not maintain the DCP, and its experimental analysis, are further studied in Chapter 8. In addition, two case studies of applying four subgraph weighting schemes (ATW, AW, CMW, and JSW) to the RT2 and RT3 data are considered in the next chapter.

Chapter 7

Two Case Studies

Two case studies illustrating the application of the four proposed subgraph weighting schemes (ATW, AW, CMW, and JSW) to two real-life data sets, RT2 (MRI brain scan) and RT3 (document trees), are presented in this chapter. The objective of these two case studies is to demonstrate that the classifiers built using patterns discovered by the weighted FSM algorithms are more effective than those built using patterns discovered by the standard FSM algorithms, in terms of both the classification accuracy and the number of features employed to build the classifiers. Thus, using a similar procedure and experimental settings as described in Chapter 6, gSpan-ATW was employed together with the SW1, SW4, and SW5 functions; gSpan-AW was employed together with the SW1 and SW4 functions; gSpan-CMW was employed together with the SW2, SW3, CW2, and CW3 functions; gSpan-JSW did not use any weighting functions.

According to the data description in Chapter 3, two-class classification problems can be formulated for RT2 and RT3. Since the efficiency and effectiveness of the classifiers using RT2 and RT3 are of main concern, the runtime cost of identifying the features, the number of features identified, and the accuracy of the classifiers are three important factors that need to be considered to determine the effectiveness of the subgraph weighting schemes. Thus, in each case study, the efficiency of using each of the four weighting schemes is discussed first, followed by discussion of the effectiveness of the patterns discovered in terms of the classification accuracy, Details of each of the case studies are presented in the following two sections.

7.1 Case Study 1 - The RT2 Data

As described in Chapter 3, the RT2 MRI brain scan data contains three data sets: QT-D5, QT-D6, and QT-D7. The three data sets represent the same collection of images but with different levels of quad-tree decomposition. The number of trees in each of the data sets is the same (106 trees). However, as indicated in Table 7.1, the trees in QT-D7 have more vertexes and edges than those in QT-D6 which have more vertexes and edges than those in QT-D5. For these data sets, only the SW1 and SW4 functions

were employed with respect to the ATW scheme. The SW5 function was not used because it could not generate meaningful vertex weights (i.e. all the vertex weights calculated by SW5 equalled zero).

Table 7.1: A summary of the RT2 data

#	Data set	# Transactions	Average $\|V(g)\|$	Average $\|E(g)\|$	$\|L_V\|$	$\|L_E\|$
1	RT2:QT-D5	106	118	117	3	4
2	RT2:QT-D6	106	150	149	3	4
3	RT2:QT-D7	106	283	282	3	4

Table 7.2: The performance of gSpan-ATW using SW1 on the RT2 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-ATW + SW1		
		gSpan	FFSM	GASTON		τ(%)	runtime	# patterns
RT2:QT-D5	30	105.793	45.426	44.838	337242	30	4.933	4877
	35	32.754	15.728	9.620	87540	35	3.688	3196
	40	15.450	8.923	6.834	36955	40	2.939	2186
	45	8.818	5.583	4.106	18448	45	2.445	1529
	50	5.061	3.881	3.136	8715	50	1.997	1128
RT2:QT-D6	35	118.280	50.243	30.360	257811	35	5.743	6430
	40	48.741	23.893	13.234	94333	40	4.156	4309
	45	24.653	13.499	7.751	41096	45	3.202	3003
	50	12.025	8.896	4.868	18353	50	2.656	2224
	55	7.630	5.913	3.745	10423	55	2.345	1631
RT2:QT-D7	45	561.675	422.965	n/a	448683	45	15.626	9214
	50	219.924	202.759	n/a	159507	50	11.838	6602
	55	56.686	68.235	n/a	39008	55	9.215	4918
	60	34.394	41.410	n/a	20905	60	6.899	3578

7.1.1 Efficiency test

Table 7.2 presents the performance of gSpan-ATW using SW1 with respect to RT2. In the table, the same range of support thresholds was used for the respective QT-D5, QT-D6, and QT-D7 data sets. It can be observed from the table that gSpan-ATW using SW1 runs significantly faster and discovers much fewer patterns than the three standard FSM algorithms. Table 7.2 further shows that using QT-D7, GASTON can not carry out the desired mining even at a support threshold as high as 60%, while gSpan-ATW with SW1 can complete the mining within a short period of time.

In the case of using SW4, the performance of gSpan-ATW on RT2, as shown in Table 7.3, also performs well with respect to the three standard FSM algorithms. However, gSpan-ATW coupled with SW4 seems to discover slightly more patterns than that coupled with SW1. Furthermore, gSpan-ATW using SW4 runs faster than when using SW1 on QT-D5, but the former runs slower than the latter on both QT-D6 and QT-D7.

An extensive analysis of gSpan-ATW using a large sequence of support thresholds on RT2, in comparison with the three standard FSM algorithms, is provided in Appendix B.1.1.2.

Although the application of gSpan-AW to the RT2 data set identifies fewer patterns

Table 7.3: The performance of gSpan-ATW using SW4 on the RT2 data

Dataset	τ(%)	gSpan-ATW + SW4	
		runtime (in seconds)	# patterns
RT2:QT-D5	30	4.428	5104
	35	3.367	3299
	40	2.642	2279
	45	2.191	1596
	50	1.947	1159
RT2:QT-D6	35	7.800	6450
	40	5.875	4383
	45	4.803	3071
	50	3.859	2239
	55	3.175	1637
RT2:QT-D7	45	25.585	10179
	50	17.560	6951
	55	14.791	5160
	60	11.192	3930

within a shorter runtime than the standard FSM algorithms, the classifiers obtained using patterns discovered by gSpan-AW tend to achieve a worse accuracy result than those built using patterns discovered by the standard FSM algorithms. This suggests that both SW1 and SW4 are not appropriate edge weightings for the trees in RT2, with respect to the AW scheme. Possible explanations for this are very similar to those discussed in Sub-section 6.4.2 for the performance of gSpan-AW using SW1 or SW4 on the RG1:CH1 data set. Accordingly, the experimental result of applying gSpan-AW on the RT2 data is omitted here (however details can be found in Appendix B.1.2).

When applying the CMW scheme to RT2, gSpan-CMW failed to work because, regardless of which of the four weighting functions (SW2, SW3, CW2, and CW3) was applied, it was not possible to produce valid edge weights (i.e. the edge weights computed using SW2, SW3, CW2, or CW3 all evaluated to zero).

In the case of the JSW scheme, the performance of gSpan-JSW with $\gamma = 0.2$ on the RT2 data is presented in Table 7.4. In the table, the same range of support thresholds was used with respect to the QT-D5, QT-D6, and QT-D7 data sets in order to show the advantage of gSpan-JSW over the standard FSM algorithms. As can be seen from the table, using $\gamma = 0.2$, gSpan-JSW mostly runs much faster and identifies significantly fewer patterns than when using the three standard FSM algorithms.

When using a different γ value, the performance of gSpan-JSW as shown in Table B.13 indicates that gSpan-JSW with a larger γ value outperforms that with a smaller γ value with respect to both runtime and the number of patterns discovered. Figure 7.1, which describes the performance comparison between three standard FSM algorithms and gSpan-JSW with two different γ values, clearly demonstrates this phenomenon.

Table 7.4: The performance of gSpan-JSW with $\gamma = 0.2$ on the RT2 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-JSW			
		gSpan	FFSM	GASTON		γ	σ(%)	runtime	# patterns
RT2:QT-D5	30	105.793	45.426	44.838	337242	0.2	30	8.482	9064
	35	32.754	15.728	9.620	87540		35	6.738	6869
	40	15.450	8.923	6.834	36955		40	5.455	5405
	45	8.818	5.583	4.106	18448		45	4.399	4306
	50	5.061	3.881	3.136	8715		50	3.503	3381
RT2:QT-D6	35	118.280	50.243	30.360	257811	0.2	35	10.825	9658
	40	48.741	23.893	13.234	94333		40	9.286	7934
	45	24.653	13.499	7.751	41096		45	7.592	6452
	50	12.025	8.896	4.868	18353		50	5.970	5194
	55	7.630	5.913	3.745	10423		55	5.158	3944
RT2:QT-D7	45	561.675	422.965	n/a	448683	0.2	45	19.036	11055
	50	219.924	202.759	n/a	159507		50	16.736	9409
	55	56.686	68.235	n/a	39008		55	14.514	7600
	60	34.394	41.410	n/a	20905		60	12.241	6183

7.1.2 Effectiveness of the patterns

The records in the RT2 data set are labelled using two classes, Musician and Control (Non-musician). Table 7.5 shows the accuracy obtained using different classifiers built using patterns discovered with both the standard FSM algorithms and gSpan-ATW coupled with SW1 when applied to the RT2 data. It can be observed from Table 7.5 that the classifiers built using patterns discovered by gSpan-ATW with SW1 achieve very close accuracy to those built using patterns discovered by the standard FSM algorithms, however the former require a considerably smaller number of features than the latter. Further, in comparison with the performance of the classifiers built using patterns discovered by gSpan-ATW with SW4, as shown in Table 7.6, it can be inferred that usage of both SW1 and SW4 have a similar effect on the accuracy of the classifiers built using gSpan-ATW patterns. However, the classifiers built using patterns discovered by gSpan-ATW with SW1 require a smaller number of features to construct the classifiers than those built using patterns discovered by gSpan-ATW with SW4.

Table 7.5: The accuracy of the classifiers using patterns discovered by gSpan-ATW with SW1 on the RT2 data

Dataset	standard FSM algorithms					gSpan-ATW + SW1				
	σ(%)	#F	NBC	SVM	C4.5	τ(%)	#F	NBC	SVM	C4.5
RT2:QT-D5	35	87540	76.4	73.6	66.0	25	8349	81.1	78.3	67.9
	40	36955	79.2	76.4	68.9	30	4877	76.4	72.6	67.0
	45	18448	73.6	69.8	56.6	35	3196	76.4	72.6	66.0
	50	8715	75.5	66.0	55.7	40	2186	77.4	74.5	70.8
RT2:QT-D6	45	41096	85.8	84.9	73.6	25	17736	85.8	85.3	78.3
	50	18353	82.1	85.8	78.3	30	10005	84.0	83.9	78.3
	55	10423	80.2	79.2	74.5	35	6430	81.1	82.1	80.2
	60	6438	81.1	80.2	73.6	40	4309	81.1	82.1	80.2
RT2:QT-D7	55	39008	82.1	77.4	62.3	55	4918	84.0	76.4	72.6
	60	20905	80.2	78.3	63.2	60	3578	85.8	78.3	76.4
	65	11998	80.2	77.4	69.8	65	2790	85.8	79.2	78.3
	70	6959	81.1	77.4	70.8	70	2103	83.0	73.6	70.8

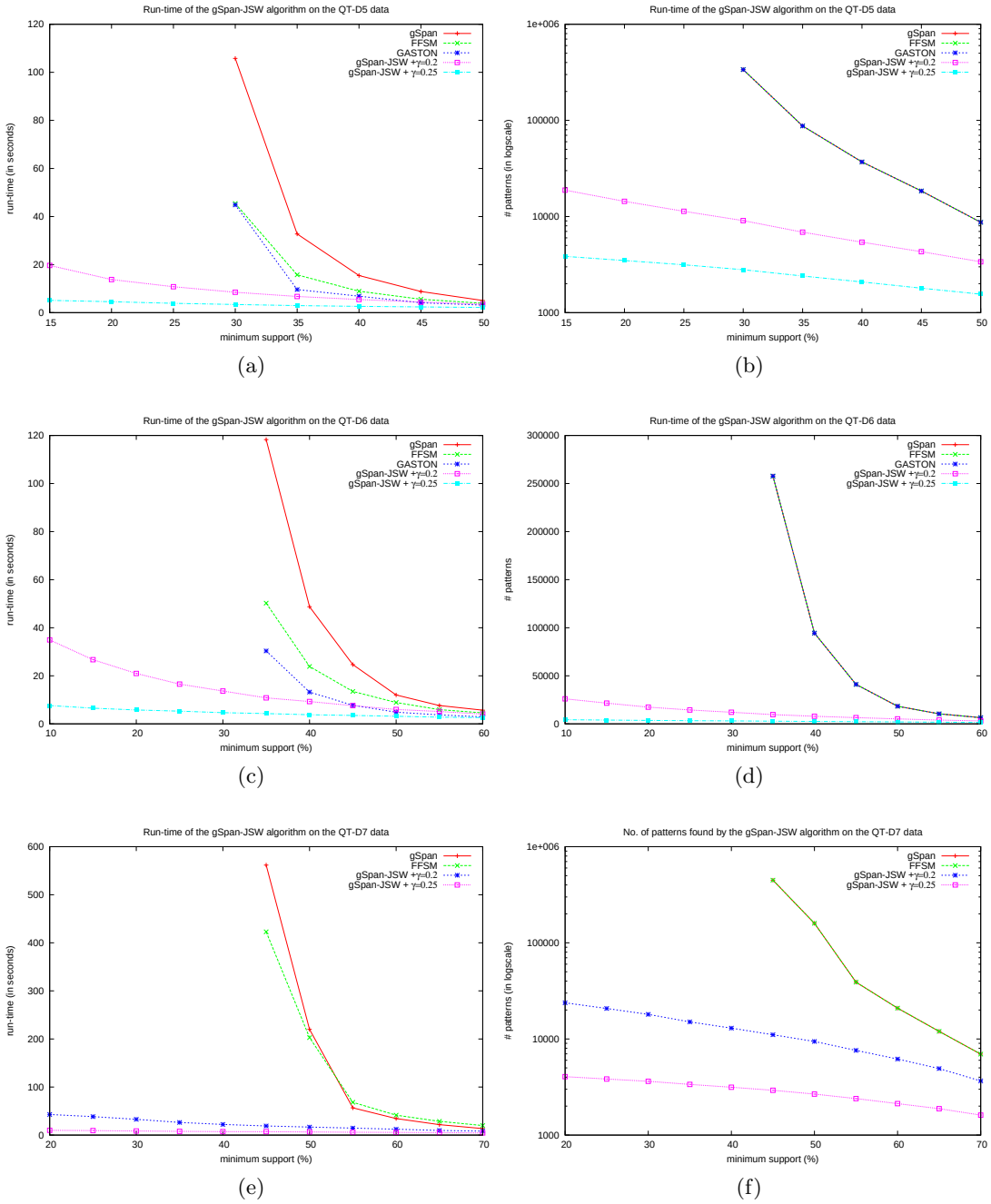


Figure 7.1: The performance of gSpan-JSW with different γ values on the RT2 data

Additionally, comparing the performance of the classifiers on the QT-D5, QT-D6, and QT-D7 data sets respectively, it appears that the performance of the classifiers on QT-D5 is worse than the performance of the classifiers on QT-D6 or QT-D7, which may suggest that the representation with a decomposition level of 5 is not very effective for the MRI brain scan images.

In the case of the JSW scheme, the patterns discovered by gSpan-JSW on the RT2 data were used to construct frequent pattern based classifiers. The accuracy results

Table 7.6: The accuracy of the classifiers using patterns discovered by gSpan-ATW with SW4 on the RT2 data

Dataset	gSpan-ATW + SW4				
	τ(%)	#F	NBC	SVM	C4.5
RT2:QT-D5	25	8749	81.1	78.3	67.9
	30	5104	76.4	72.6	67.0
	35	3299	76.4	72.6	66.0
	40	2279	77.4	74.5	70.8
RT2:QT-D6	25	18065	85.8	85.8	78.3
	30	10251	84.0	83.9	78.3
	35	6450	81.1	82.1	80.2
	40	4383	81.1	82.1	80.2
RT2:QT-D7	55	5160	84.0	76.4	72.6
	60	3930	85.8	78.3	75.5
	65	2918	85.8	79.2	78.3
	70	2214	85.8	75.5	76.4

obtained are shown in Table 7.7. In the table, different ranges of support thresholds were used for the respective standard FSM algorithms and gSpan-JSW to extract effective patterns for the classification. Compared with the result of the classifiers built using pattern discovered using the standard FSM algorithms, the accuracy of the NBC and C4.5 classifiers using patterns discovered by gSpan-JSW is higher than that using patterns discovered by the standard FSM algorithms, and the accuracy of the SVM classifier using patterns discovered by gSpan-JSW is very close to that using pattern discovered by the standard FSM algorithms. On the whole, the classifiers built using patterns discovered by gSpan-JSW employ far fewer features than those when using the standard FSM algorithms.

Table 7.7: The accuracy of the classifiers using patterns discovered by gSpan-JSW with different γ values on the RT2 data

Dataset	standard FSM algorithms					vs.	gSpan-JSW					
	σ(%)	#F	NBC	SVM	C4.5		γ	σ(%)	#F	NBC	SVM	C4.5
RT2:QT-D5	35	87540	76.4	73.6	66.0		0.2	15	18820	88.7	72.6	79.2
	40	36955	79.2	76.4	68.9			20	14376	86.8	73.6	83.0
	45	18448	73.6	69.8	56.6		0.25	2	4650	85.8	69.8	71.7
	50	8715	75.5	66.0	55.7			5	4473	82.1	70.8	79.2
RT2:QT-D6	45	41096	85.8	84.9	73.6		0.2	15	21591	90.6	82.1	82.1
	50	18353	82.1	85.8	78.3			20	17431	85.8	75.5	77.4
	55	10423	80.2	79.2	74.5		0.25	2	4732	87.7	74.5	78.3
	60	6438	81.1	80.2	73.6			5	4552	86.8	73.6	78.3
RT2:QT-D7	55	39008	82.1	77.4	62.3		0.2	25	20730	84.9	78.3	76.4
	60	20905	80.2	78.3	63.2			30	18022	84.0	75.5	75.5
	65	11998	80.2	77.4	69.8		0.25	2	4886	82.1	74.5	69.8
	70	6959	81.1	77.4	70.8			5	4744	82.1	75.5	72.6

Moreover, Table 7.7 also suggests that the classifiers obtained using patterns discovered by gSpan-JSW with a larger γ value achieve a slightly worse performance than those obtained using patterns discovered by gSpan-JSW with a smaller γ value. However, using a larger γ value, the number of features required to build the classifiers is

significantly less than when using a smaller γ value.

7.1.3 Summary

As discussed in the last two sub-sections, two subgraph weighting schemes: ATW and JSW were applicable to the RT2 data. In comparison with using patterns discovered by the standard FSM algorithms, the classifiers built using patterns discovered by gSpan-ATW or gSpan-JSW achieve very close performance to those built using patterns discovered by the standard FSM algorithms but the former employ considerably fewer features than the latter. More importantly, the runtime cost of identifying such a smaller number of features by gSpan-ATW or gSpan-JSW is significantly less than that when using the standard FSM algorithms. Further, according to the performance of the classifiers on the respective QT-D5, QT-D6, and QT-D7 data sets, as shown in Tables 7.5, 7.6, 7.7, QT-D6 seems to be an optimum tree representation for the brain scan data. It should also be noted that a complete study of applying gSpan-ATW to the RT2 MRI brain scan data, in the perspective of medical image classification, can be found in Elsayed et al. [2010].

7.2 Case Study 2 - The RT3 Data

This section presents a study of the application of the four proposed subgraph weighting schemes to the RT3 document trees data. The properties of the RT3 data are summarized in Table 7.8. Among the four schemes, recall that JSW does not use vertex or edge weightings, the appropriate weighting functions introduced in Chapter 4 were therefore employed, in conjunction with the respective ATW, AW, and CMW subgraph schemes. More specifically, ATW employs the SW1, SW4, and CW1-N functions; AW employs the SW1 and SW4 functions; CMW employs the CW2 and CW3 functions. Thus, the evaluation of these subgraph weighting schemes on RT3 is examined in the subsequent sub-sections, according to the efficiency of the corresponding weighting scheme and the effectiveness of the patterns discovered by applying the corresponding weighting scheme.

Table 7.8: A summary of the RT3 data

Data set	# Transactions	Average $\|V(g)\|$	Average $\|E(g)\|$	$\|L_V\|$	$\|L_E\|$
RT3	200	1142	1141	10069	6

7.2.1 Efficiency test

When the standard FSM algorithms, FFSM and GASTON, were applied to the RT3 data set it was discovered that both FFSM and GASTON were unable to operate using even very high support thresholds, such as 90%, due to out-of-memory errors.

Further tests of using gSpan-ATW, indicated that gSpan-ATW can accommodate the RT3 data better than the FFSM and GASTON algorithms. Thus, the performance of gSpan-ATW with SW1, SW4 and CW1-N, in comparison with gSpan, is illustrated in Figure 7.2. From the figure it can be seen that the curve representing gSpan stops at a support threshold of 80%, again due to an out-of-memory error, while gSpan-ATW continues down to a support threshold of 60%. Generally, Figure 7.2 indicates that gSpan-ATW using CW1-N seems to perform moderately better than that using SW1 or SW4. This fact may suggest that for the RT3 data, the user provided weighting is more accurate than the structure based weighting functions. Overall it can be inferred that the ATW scheme is not applicable to the RT3 data, given the performance of both standard FSM algorithms and gSpan-ATW. Consequently no further evaluation with respect to the ATW scheme was conducted.

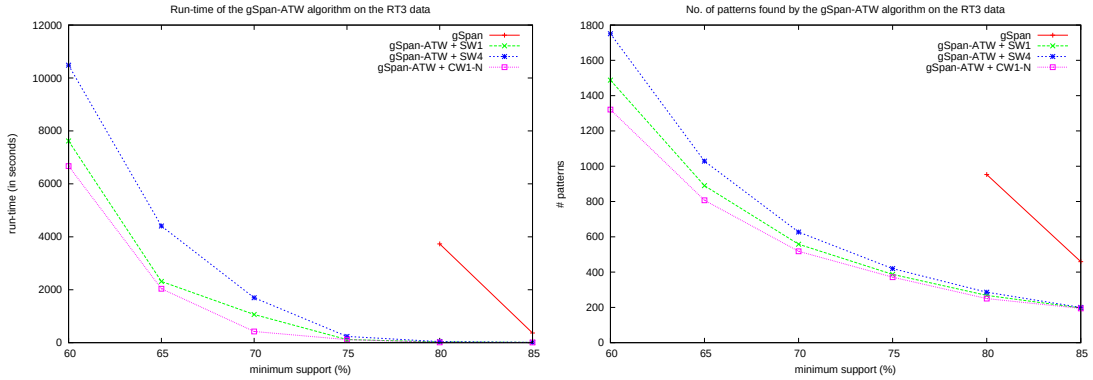


Figure 7.2: The performance of gSpan-ATW on the RT3 data

Since the standard FSM algorithms and gSpan-ATW failed to complete the mining task when applied to RT3, even with a relatively high support threshold, only the performance of gSpan-AW, gSpan-CMW, and gSpan-JSW on RT3 is described in the following paragraphs.

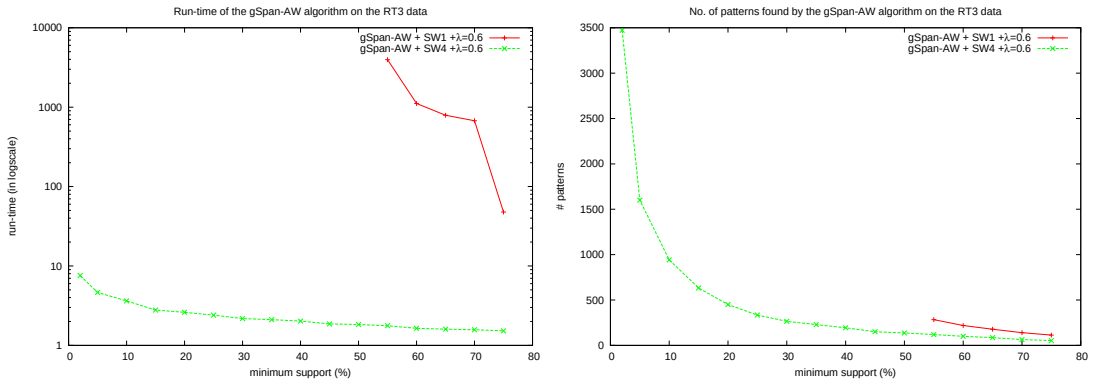
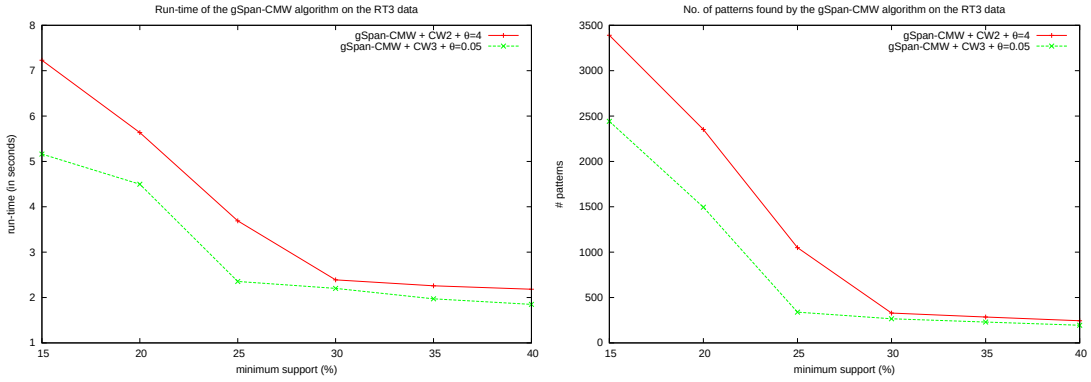


Figure 7.3: The performance of gSpan-AW on the RT3 data

Figure 7.3 compares the performance of gSpan-AW coupled with SW1 with that of gSpan-AW using SW4. As can be seen clearly from the figure, gSpan-AW coupled with SW4 requires the least amount of runtime to identify a relatively small number of patterns, while gSpan-AW coupled with SW1 can not finish the mining when the support threshold is decreased to below 55% due to an out-of-memory error. This remarkably large difference in runtime cost between gSpan-AW using SW1 and gSpan-AW using SW4 clearly indicates that the SW1 weighting function is not an appropriate edge weighting mechanism for the RT3 data set with respect to the AW scheme. Further inspection of the mining process of gSpan-AW using SW1 on RT3 reveals that the reason for the high runtime cost incurred by gSpan-AW using SW1 is that a large proportion of the weighting ratio values computed using SW1 were actually equal to 1, which causes the pruning strategy based on the weighting ratio to be ineffective.



(a) (b)

Figure 7.4: The performance of gSpan-CMW on the RT3 data

When using the CMW scheme, Figure 7.4 presents the performance of gSpan-CMW coupled with either CW2 or CW3. As can be seen from the figure, gSpan-CMW coupled with either CW2 or CW3 can identify a relatively small number of patterns within a short period of time. Furthermore, gSpan-CMW coupled with CW3 appears to run faster and discover fewer patterns than when using CW2.

In the case of the JSW scheme, the performance of gSpan-JSW is presented in Figure 7.5. In the figure the two curves representing the performance of gSpan-JSW, using different γ values, are very similar when the support thresholds are between 4% and 8%; when the support threshold is below 4%, the distinction between the two becomes significant. It appears that gSpan-JSW using a higher γ value performs better than that using a lower γ value, in terms of both runtime cost and the number of discovered patterns.

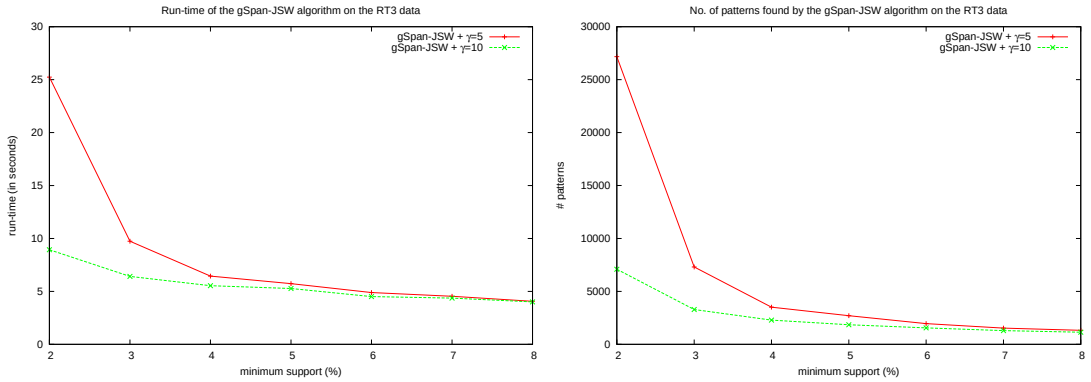


Figure 7.5: The performance of gSpan-JSW on the RT3 data

7.2.2 Effectiveness of the patterns

The patterns discovered by the respective gSpan-AW, gSpan-CMW, and gSpan-JSW algorithms when applied to the RT3 data were used to construct frequent pattern based classifiers. The accuracy of the classifiers obtained is thus used to determine the effectiveness of the patterns discovered by the corresponding weighted FSM algorithm.

Since gSpan-AW coupled with SW1 was not efficient at discovering patterns as discussed before, only the patterns discovered by gSpan-AW coupled with SW4 were employed for classification purposes. The accuracy results obtained are shown in Table 7.9. As can be seen from the table, the classifiers achieved good results with a relatively small number of features.

Table 7.9: The accuracy of the classifiers using patterns discovered by the weighted FSM algorithms on the RT3 data

gSpan-AW + SW4						vs.	gSpan-JSW					
λ	τ(%)	#F	NBC	SVM	C4.5		γ	σ(%)	#F	NBC	SVM	C4.5
0.6	5	1600	98.5	83.5	89.5		5	5	2702	99.0	87.0	90.0
	10	942	97.5	88.5	94.0			6	1956	98.0	86.0	92.0
	15	633	95.5	78.5	91.5			7	1532	98.5	83.5	93.0
gSpan-CMW + CW2						vs.	gSpan-CMW + CW3					
θ	σ(%)	#F	NBC	SVM	C4.5		θ	σ(%)	#F	NBC	SVM	C4.5
4	15	3390	96.0	80.5	91.5		0.05	15	2442	95.5	80.5	91.5
	20	2352	94.0	76.5	90.0			20	1495	93.5	77.0	93.0
	25	1050	92.5	77.0	91.5			25	339	92.5	76.5	91.5

Table 7.9 further reveals that the accuracy of the classifiers built using patterns discovered by gSpan-CMW coupled with either CW2 or CW3 is rather good. Furthermore, the classifiers built using patterns discovered by gSpan-CMW coupled with CW3 required far fewer patterns than those built using CW2 to achieve very similar results.

The classification results using gSpan-JSW are also shown in Table 7.9. As can be seen from the table, the accuracy of each classifier is very good. Additionally it should be noted that this high accuracy is achieved using a relatively small number of features

and good runtime performance (see Figure 7.5).

Altogether, in comparison with gSpan-JSW, gSpan-CMW, Table 7.9 also suggests that the classifiers built using gSpan-AW coupled with SW4 use the smallest number of features to obtain a similar high level of accuracy.

7.2.3 Summary

The advantage of using weighted FSM algorithms over the standard FSM algorithms is effectively demonstrated by their performance on the RT3 data. Because the standard FSM algorithms failed to operate on the RT3 data, effective patterns could not be identified in order to adopt the framework of the frequent pattern based classification to classify the RT3 data. Under this situation, four weighted FSM algorithms: gSpan-ATW, gSpan-AW, gSpan-CMW, and gSpan-JSW were employed to extract the patterns suitable for the classification from the RT3 data. Although gSpan-ATW was not applicable to the RT3 data because of out-of-memory errors, the other three algorithms: gSpan-AW, gSpan-CMW, and gSpan-JSW identify a relatively small number of patterns within a very short period of runtime. More importantly, the classifiers built using such patterns discovered by the weighted FSM algorithms are capable of obtaining the desired high accuracy. An alternative approach to applying weighted FSM algorithms to the RT3 document data can be found in Jiang et al. [2010a].

Chapter 8

Subgraph Weighting Schemes That Do Not Maintain the DCP

As introduced in Chapter 5, the DCP plays an important role in reducing the computation effort in the process of FSM. Thus, it is desirable to devise a subgraph weighting scheme that maintain the DCP. An alternative approach is to identify subgraph weighting schemes that limit the growth of the search space in some other way. When deriving such weighting schemes care needs to be taken to ensure that, whatever alternative to maintaining the DCP that is adopted, it is still efficient and effective. The adopted scheme should be more efficient than when no weightings are used. At the same time such weighting schemes should be effective; the correct patterns should still be identified. Since it is hard to devise an efficient and effective subgraph weighting scheme that does not maintain the DCP, only one subgraph weighting scheme, Utility Based Weighting (UBW), is described in this chapter. Again effectiveness is measured in terms of classification accuracy when the identified patterns are used to build the classifiers.

8.1 Utility Based Weighting (UBW) Scheme

The formulation of the UBW scheme is influenced by ideas suggested in Carter et al. [1997] and Barber and Hamilton [2003]. It is founded on two components: (i) weighted support and (ii) the share (SH) of a subgraph [Jiang et al., 2010c]. Thus:

Definition 8.1.1. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$ and a subgraph g with edges $E(g) = \{e_1, e_2, \cdots, e_k\}$, let two vertexes connecting each $e_i \in E(g)$ equal v_1, v_2, and let the set of graphs where v_1 occurs equal $\Gamma_{\mathbb{GD}}(v_1)$ and set of graphs where v_2 occurs equal $\Gamma_{\mathbb{GD}}(v_2)$. Then the overlap similarity between two vertexes is defined as:*

$$S_{overlap}(e_i) = \frac{|\Gamma_{\mathbb{GD}}(v_1) \cap \Gamma_{\mathbb{GD}}(v_2)|}{min(|\Gamma_{\mathbb{GD}}(v_1)|, |\Gamma_{\mathbb{GD}}(v_2)|)} \quad . \tag{8.1}$$

Thus, the weight of g, $W_{\mathbb{GD}}(g)$, is then defined as

$$W_{\mathbb{GD}}(g) = \frac{1}{\sum_{e_i \in E(g)} S_{overlap}(e_i)} \ . \tag{8.2}$$

Equation 8.1 can also be replaced with other similarity measures such as Jaccard similarity, Dice similarity and Cosine similarity. Accordingly, the weighted support is given by $wsup_{\mathbb{GD}}(g) = sup_{\mathbb{GD}}(g) \times W_{\mathbb{GD}}(g)$.

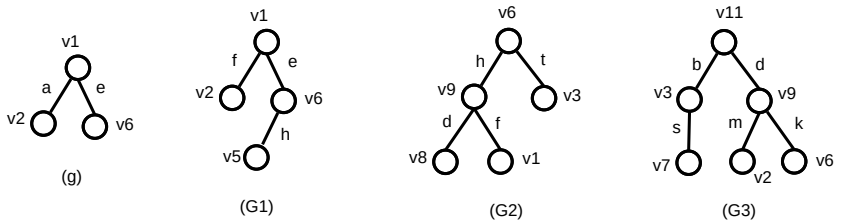


Figure 8.1: An example of computing the overlap similarity

Example: Considering the graph data set $\mathbb{GD} = \{G1, G2, G3\}$ given in Figure 8.1, where the symbol next to each vertex or edge represents the label. Given a subgraph g which contains two edges $\{a, e\}$, $\Gamma_{\mathbb{GD}}(v1) = \{G1, G2\}$, $\Gamma_{\mathbb{GD}}(v2) = \{G1, G3\}$, and $\Gamma_{\mathbb{GD}}(v6) = \{G1, G2, G3\}$. Then, $S_{overlap}(a) = 0.5$, $S_{overlap}(e) = 1$, and $W_{\mathbb{GD}}(g) = \frac{1}{1+0.5} \approx 0.67$.

In order to delineate the concept of the share of a subgraph, a sequence of definitions are firstly introduced in the following paragraphs.

Definition 8.1.2. *Given an edge weighted graph set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$ with edge weights $\{w_1, w_2, \cdots, w_k\}$ for each graph G_j, let the set of graphs where g occurs equal $\delta_{\mathbb{GD}}(g)$. Then the subgraph weight of g in each transaction graph G_j, denoted as $W_{G_j}(g)$, is the sum of the weight of each edge in g occurring in G_j:*

$$W_{G_j}(g) = \sum_{e_i \in g \cap g \subseteq_{sub} G_j} w_i \ . \tag{8.3}$$

The total weight of $\mathbb{GD}$, denoted as $TW(\mathbb{GD})$, represents the sum of all edge weights in $\mathbb{GD}$. That is:

$$TW(\mathbb{GD}) = \sum_{G_j \in \mathbb{GD}} \sum_{e_i \in G_j} w_i \ . \tag{8.4}$$

In addition, the total weight of $\delta_{\mathbb{GD}}(g)$ is defined as:

$$TW(\delta_{\mathbb{GD}}(g)) = \sum_{G_j \in \delta_{\mathbb{GD}}(g)} \sum_{e_i \in G_j} w_i \ . \tag{8.5}$$

Definition 8.1.3. *The graph weight of g with respect to $\mathbb{GD}$, denoted as $GW_{\mathbb{GD}}(g)$, is the sum of the subgraph weight of g in each graph $G_j \in \delta_{\mathbb{GD}}(g)$. That is:*

$$GW_{\mathbb{GD}}(g) = \sum_{G_j \in \delta_{\mathbb{GD}}(g)} W_{G_j}(g) \ . \tag{8.6}$$

Definition 8.1.4. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, the share of a subgraph g with respect to $\mathbb{GD}$, denoted as $SH_{\mathbb{GD}}(g)$, is the ratio of the graph weight of g with respect to $\mathbb{GD}$ to the total weight of $\mathbb{GD}$. Thus:*

$$SH_{\mathbb{GD}}(g) = \frac{GW_{\mathbb{GD}}(g)}{TW(\mathbb{GD})} \ . \tag{8.7}$$

Given a share threshold μ, a subgraph g is SH-frequent if $SH_{\mathbb{GD}}(g) \geq \mu$; otherwise, g is SH-infrequent.

It can be easily inferred form (8.7) that the SH measure does not maintain the DCP. Therefore, it is important to utilize some properties of the SH measure to define some pruning strategy, which can be employed to remove predicted infrequent pattens from the search space. Thus the following theorem is introduced.

Theorem 8.1.1. *Given a graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, a subgraph g, and a threshold μ, if*

$$w_r = \frac{TW(\delta_{\mathbb{GD}}(g))}{TW(\mathbb{GD})} < \mu \ .$$

then all supergraphs of g are SH-infrequent.

Proof. Let h be an arbitrary supergraph of g. Clearly, $GW_{\mathbb{GD}}(h) \leq TW(\delta_{\mathbb{GD}}(h)) \leq TW(\delta_{\mathbb{GD}}(g))$. If $TW(\delta_{\mathbb{GD}}(g)) < \mu \times TW(\mathbb{GD})$ holds, then $GW_{\mathbb{GD}}(h) < \mu \times TW(\mathbb{GD})$. That is, $SH_{\mathbb{GD}}(h) = \frac{GW_{\mathbb{GD}}(h)}{TW(\mathbb{GD})} < \mu$. Therefore, h is SH-infrequent. □

Definition 8.1.5. *Given an edge weighted graph data set $\mathbb{GD} = \{G_1, G_2, \cdots, G_n\}$, a weighted support threshold $\tau \in (0, 1]$, and a share threshold $\mu \in (0, 1]$, a subgraph g is a weighted frequent pattern if the following two conditions are satisfied:*

$$\textbf{(D1)}\ wsup_{\mathbb{GD}}(g) \geq \tau, \qquad \textit{and} \qquad \textbf{(D2)}\ SH_{\mathbb{GD}}(g) \geq \mu \quad .$$

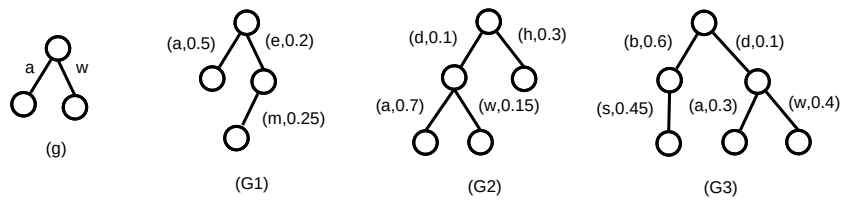


Figure 8.2: An example of computing share values.

Example: Considering the graph data set $\mathbb{GD} = \{G1, G2, G3\}$ shown in Figure 8.2, where the symbol next to each edge, '(a,b)', indicates the edge label (denoted by 'a') and the weight (denoted by 'b'); vertex labels are not included. Given a subgraph

g which occurs in $\{G2, G3\}$, $TW(\mathbb{GD}) = 0.5 + 0.2 + 0.25 + 0.1 + 0.3 + 0.7 + 0.15 + 0.6 + 0.1 + 0.45 + 0.3 + 0.4 = 4.05$, $GW_{\mathbb{GD}}(g) = 0.7 + 0.15 + 0.3 + 0.4 = 1.55$. So, $SH_{\mathbb{GD}}(g) = 1.55/4.05 \approx 0.38$.

Algorithm 8.1: subgSpan-UBW(c, $\mathbb{GD}$, τ, μ, $\mathcal{F}$)

```
1  if c ≠ min(c) then
2  |  return
3  end
4  if wsup_GD(c) ≥ τ ∧ SH_GD(c) ≥ μ then
5  |  F ← F ∪ {c}
6  else if wsup_GD(c) ≥ τ then
7  |  if w_r = TW(δ_GD(c)) / TW(GD) ≥ μ then
8  |  |  subgSpan-UBW(c, GD, τ, μ, F)
9  |  else
10 |  |  return
11 |  end
12 end
13 C ← ∅
14 Scan GD once, find every edge e such that c can be right-most extended to c ∪ e,
   C ← c ∪ e
15 Sort C in DFS lexicographic order
16 foreach g_k ∈ C do
17 |  if wsup_GD(g_k) ≥ τ ∧ SH_GD(g_k) ≥ μ then
18 |  |  subgSpan-UBW(g_k, GD, τ, μ, F)
19 |  else if wsup_GD(g_k) ≥ τ then
20 |  |  if w_r = TW(δ_GD(g_k)) / TW(GD) ≥ μ then
21 |  |  |  subgSpan-UBW(g_k, GD, τ, μ, F)
22 |  |  end
23 |  end
24 end
```

8.1.1 Pseudo-codes of UBW

The UBW scheme was implemented by incorporating it into gSpan to give gSpan-UBW. Algorithm 8.1 displays the key elements of the UBW scheme. From lines 4 to 12, and lines 16 to 24, if both conditions **(D1)** and **(D2)** are satisfied, the subgraph candidate g_k will become a weighted frequent pattern; if condition **(D1)** holds and condition **(D2)** does not hold, because the SH measure does not maintain the DCP, the value of w_r in Theorem 8.1.1 is computed firstly during the mining process. Under this situation, if $w_r < \mu$, then all the supergraphs of g_k (including g_k) are SH-infrequent and can be safely pruned from that branch of the search space; otherwise, the candidate subgraph g_k will be considered at the next iteration, because one or more of the supergraphs of g_k may also be frequent. Therefore, Theorem 8.1.1 plays an important role in reducing

the search space during the mining operation. Practically, the effectiveness of the UBW scheme is dependent on the pruning capability facilitated by Theorem 8.1.1. During the mining, if the number of cases, where the value of w_r is less than the share threshold, is large; the pruning power when using Theorem 8.1.1 is effective. However, if the number of cases, where the value of w_r is less than the share threshold, is small, the pruning power when using Theorem 8.1.1 is weak. Therefore, it is important to choose an appropriate edge weighting, in order to maximise the pruning power when using the UBW scheme.

8.2 Experimental Study

The evaluation of the UBW scheme, using various data sets employed by other subgraph weighting schemes, is reported in this section. The three standard FSM algorithms employed previously, gSpan, FFSM and GASTON were again used for comparison purposes. However, similar to Chapter 6, because FFSM and GASTON, when used with respect to some data sets, encountered the out-of-memory errors (the maximum memory usage is 3GB), the performance of these two algorithms was not always reported. Again, following the implementation details introduced in Section 6.1, gSpan was chosen as a base algorithm such that the UBW scheme was integrated into gSpan by the author to create the gSpan-UBW algorithm. Both gSpan and gSpan-UBW were modified by the author to accommodate directed graphs. All the experimental settings described in Chapter 6 will be adopted for the test of the UBW scheme unless otherwise stated.

The UBW scheme requires edge weights, thus all the data sets used for the evaluation of the UBW scheme can be divided into two groups, according to whether the data sets contain edge weights or not.

- **Division A:** Data sets that do not have edge weights and thus such weightings must be derived: ST1 (synthetic trees), ST2 (synthetic images), RT1 (Web logs), RT2 (MRI brain scan), RT3 (document trees), and RG1 (Chemical compounds).
- **Division B:** Data sets that do have predefined edge weights: RG2 (Mammography), RG3 (photographic images), RG4 (document base), and RG5 (social network)

For data sets that belong to Division A, due to the reason described in Chapter 5, only two structural weighting functions: SW1 and SW4 were used to generate the desired edge weights so that the UBW scheme could be applied to these data sets. However, it should be noted that by deriving edge weightings in this manner there is no guarantee that the weightings actually reflect the application domain in each case. The only motivation for using these two structural weighting functions is that the user

provided edge weights are not available, the application of SW1 and SW4 is assumed to give an appropriate weighting. For data sets belonging to Group B, the content weighting function, CW1-E, was used for the implementation of the UBW scheme. The detailed evaluation of the UBW scheme, as applied to each data set, is described in the following sub-sections, in the order of trees, undirected graphs and directed graphs.

8.2.1 The evaluation of the UBW scheme on trees

As mentioned in the above, two weighting functions: SW1 and SW4 were used to generate appropriate weights for applying the UBW scheme to ST1, ST2, RT1, RT2 and RT3.

Efficiency test. The performance of gSpan-UBW with SW1 on the ST1, ST2, RT1, RT2, and RT3 data is displayed in Table 8.1. In the table, the symbols μ and τ (lines 7 to 8) denote the share and the support thresholds used in the UBW scheme respectively (these symbols will keep the same meaning throughout the rest of this chapter). From the table, it can be seen that gSpan-UBW cuts down enormously on the number of patterns discovered by the standard FSM algorithms. However, the runtime cost for each group of data varies. Specifically, for the ST1 and RT1 data, gSpan-UBW with SW1 requires more runtime than the standard FSM algorithms while for the ST2, RT2, and RT3 data, gSpan-UBW with SW1 runs significantly faster than the standard FSM algorithms.

In the case of SW4, Table 8.2 shows that the performance of gSpan-UBW using SW4 on the ST1, ST2, and RT1 is similar to that of gSpan-UBW with SW1, and gSpan-UBW with SW4 runs slightly slower and identifies fewer patterns than gSpan-UBW with SW1 on the RT2 and RT3 data. An extended analysis of gSpan-UBW with SW1 and SW4 using a wide range of support thresholds on the RT1 and RT2 data, in comparison with the standard FSM algorithms, can also be found in Appendix B.1.5.

Classification evaluation. Since the trees in the ST2, RT1:CSLOGS-1(2), RT2, and RT3 data have class labels, the nature of the patterns discovered by gSpan-UBW could be evaluated using a classification scenario. The accuracy results for the classifiers built using patterns discovered by gSpan-UBW when applied to the ST2, RT1:CSLOGS-1(2), RT2, and RT3 are presented in Table 8.3. For the ST2 and RT1:CSLOGS-1(2) data, Table 8.3 reveals that the accuracy of the classifiers built using patterns discovered by gSpan-UBW is marginally higher than that of the classifiers built using patterns discovered by the standard FSM algorithms. More importantly, the gSpan-UBW classifiers require far fewer patterns than the classifiers built using patterns discovered by the standard FSM algorithms.

For the RT2 data, it can be seen in the table that the performance of the classifiers built using patterns discovered by gSpan-UBW with SW1 is close to that of the

Table 8.1: The performance of gSpan-UBW with SW1 on the tree data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-UBW + SW1			
		gSpan	FFSM	GASTON		μ	τ(%)	runtime	# patterns
ST1:D10	0.05	10.102	8.657	14.037	21057	0.1	0.05	31.452	97
	0.1	7.586	5.927	9.473	7257		0.1	24.129	77
	0.5	4.649	3.465	6.322	727		0.5	11.598	32
	1	3.913	2.743	4.917	196		1	10.064	24
	2	3.341	2.556	4.782	179		2	9.454	23
ST1:T1M	0.05	183.656	167.000	n/a	24492	0.1	0.05	1150.950	81
	0.1	135.357	115.810	n/a	9493		0.1	921.150	63
	0.5	71.071	51.535	109.163	607		0.5	414.124	31
	1	60.592	36.970	90.239	196		1	365.927	24
	2	59.691	34.053	84.725	178		2	321.038	22
ST2:IM1000-D4	6	142.697	77.36	n/a	435890	0.2%	6	12.635	421
	8	19.815	10.143	15.752	41398		8	6.552	270
	10	12.146	6.161	9.778	21967		10	3.959	179
	12	5.395	3.067	5.737	7263		12	2.745	128
	14	3.100	1.940	4.638	2782		14	1.882	98
ST2:IM1000-D5	10	563.748	304.526	n/a	489352	0.2%	10	31.127	469
	12	110.368	67.821	51.791	61333		12	19.938	309
	14	66.628	43.268	33.616	31460		14	13.889	248
	16	42.970	30.006	20.871	16399		16	9.290	183
	18	28.725	22.882	15.286	9215		18	6.237	123
ST2:IM1000-D6	15	592.118	407.344	n/a	112683	0.2%	15	37.060	376
	20	308.146	233.924	n/a	49289		20	13.727	164
	25	132.038	109.170	44.885	14644		25	7.296	84
	30	69.509	67.888	27.517	6936		30	4.560	63
RT1:CSLOGS-ALL	0.3	4.780	3.902	9.920	2268	2%	0.3	17.810	624
	0.4	3.358	2.867	7.319	1134		0.4	11.586	433
	0.5	2.465	2.543	6.650	684		0.5	9.695	312
	0.6	2.311	2.321	5.837	498		0.6	8.364	252
	0.8	2.063	2.062	5.528	311		0.8	6.652	176
RT1:CSLOGS-1	0.2	9.871	3.189	5.117	46104	0.5%	0.2	4.914	645
	0.4	1.267	0.637	2.050	2582		0.4	3.245	303
	0.6	0.753	0.455	1.789	833		0.6	2.324	189
	0.8	0.623	0.373	1.667	455		0.8	1.919	147
	1	0.554	0.334	1.561	286		1	1.482	118
RT1:CSLOGS-2	0.2	6.388	2.503	3.633	41601	0.5%	0.2	4.072	690
	0.4	1.080	0.592	1.790	2485		0.4	2.870	314
	0.6	0.679	0.430	1.571	836		0.6	2.075	196
	0.8	0.540	0.350	1.444	462		0.8	1.731	151
	1	0.495	0.299	1.349	280		1	1.451	118
RT2:QT-D5	30	105.793	45.426	44.838	337242	0.8%	10	7.836	4498
	35	32.754	15.728	9.620	87540		15	2.768	1352
	40	15.450	8.923	6.834	36955		20	1.401	587
	45	8.818	5.583	4.106	18448		25	0.873	282
	50	5.061	3.881	3.136	8715		30	0.739	165
RT2:QT-D6	35	118.280	50.243	30.360	257811	0.8%	10	14.133	6986
	40	48.741	23.893	13.234	94333		15	4.744	1963
	45	24.653	13.499	7.751	41096		20	2.315	776
	50	12.025	8.896	4.868	18353		25	1.143	328
	55	7.630	5.913	3.745	10423		30	0.906	200
RT2:QT-D7	45	561.675	422.965	n/a	448683	0.4%	15	11.689	3512
	50	219.924	202.759	n/a	159507		20	4.878	1195
	55	56.686	68.235	n/a	39008		25	1.916	367
	60	34.394	41.410	n/a	20905		30	1.619	262
RT3	15	n/a				0.2%	15	119.116	1586
	20						20	16.806	840
	25						25	6.423	531
	30						30	4.676	380

classifiers built using patterns discovered by the standard FSM algorithms. Again, the number of features required by the classifiers built using patterns discovered by

Table 8.2: The performance of gSpan-UBW with SW4 on the tree data

Dataset	μ	τ(%)	gSpan-UBW + SW4	
			runtime (in seconds)	# patterns
ST1:D10	0.1	0.05	33.515	99
		0.1	24.920	79
		0.5	11.818	34
		1	10.499	26
		2	9.497	25
ST1:T1M	0.1	0.05	1156.202	83
		0.1	867.491	65
		0.5	415.598	33
		1	364.959	26
		2	320.328	24
ST2:IM1000-D4	0.2%	6	12.672	421
		8	6.607	270
		10	4.020	179
		12	2.686	128
		14	1.905	98
ST2:IM1000-D5	0.2%	10	29.940	469
		12	19.594	309
		14	13.475	248
		16	9.328	183
		18	6.165	123
ST2:IM1000-D6	0.2%	15	37.823	376
		20	13.794	164
		25	7.457	84
		30	4.527	63
RT1:CSLOGS-ALL	2%	0.3	22.824	646
		0.4	14.794	456
		0.5	12.326	335
		0.6	10.022	270
		0.8	7.554	194
RT1:CSLOGS-1	0.5%	0.2	5.148	650
		0.4	3.385	312
		0.6	2.418	197
		0.8	1.825	154
		1	1.545	125
RT1:CSLOGS-2	0.5%	0.2	4.602	694
		0.4	2.964	322
		0.6	2.199	204
		0.8	1.731	159
		1	1.513	126
RT2:QT-D5	0.8%	10	12.722	4450
		15	3.904	1331
		20	1.856	577
		25	1.044	276
		30	0.781	159
RT2:QT-D6	0.8%	10	22.549	6713
		15	6.451	1900
		20	2.893	734
		25	1.415	304
		30	1.110	189
RT2:QT-D7	0.4%	15	15.349	3225
		20	6.021	1099
		25	2.297	286
		30	1.833	207
RT3	0.2%	15	132.277	1539
		20	17.608	812
		25	6.816	515
		30	4.713	370

gSpan-UBW with SW1 is noticeably less than that required by the classifiers built using patterns discovered by the standard FSM algorithms.

Table 8.3: The accuracy of the classifiers using patterns discovered by gSpan-UBW using SW1 on the tree data

Dataset	standard FSM algorithms					vs.	gSpan-UBW + SW1					
	σ(%)	#F	NBC	SVM	C4.5		μ	τ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	12	7263	92.9	95.4	94.9		0.2%	6	421	93.3	95.4	93.8
	14	2782	92.7	95.4	94.6			8	270	93.5	95.3	93.9
	16	1659	92.6	95.5	94.3			10	179	92.3	95.7	94.4
ST2:IM1000-D5	20	6287	87.0	91.4	91.9		0.2%	6	1573	90.3	92.9	92.3
	25	2453	86.2	91.4	91.3			8	787	86.1	92.0	91.9
	30	1163	86.0	91.4	70.8			10	469	84.3	91.9	90.0
ST2:IM1000-D6	30	6936	81.9	76.4	86.8		0.2%	8	1972	85.9	80.3	90.0
	35	3720	81.2	75.1	86.5			10	1113	85.2	79.3	89.8
	40	1869	81.0	75.2	86.7			12	646	83.6	76.2	88.8
RT1:CSLOGS-1	0.3	7971	79.8	81.8	81.8		0.5%	0.1	1298	80.6	81.2	82.1
	0.4	2582	79.8	81.8	81.8			0.2	645	80.4	81.8	82.1
	0.5	1286	79.8	81.2	81.2			0.4	303	80.6	81.7	82.1
RT1:CSLOGS-2	0.3	6445	80.4	81.9	82.1		0.5%	0.1	1353	80.6	80.7	82.4
	0.4	2485	80.4	82.0	82.1			0.2	690	80.6	81.3	82.2
	0.5	1281	80.4	81.9	81.9			0.4	314	80.6	82.4	82.2
RT2:QT-D5	35	87540	76.4	73.6	66.0		0.8%	8	8856	81.1	72.6	73.6
	40	36955	79.2	76.4	68.9			10	4498	75.5	70.8	69.8
	45	18448	73.6	69.8	56.6			12	2591	73.6	70.8	76.4
RT2:QT-D6	45	41096	85.8	84.9	73.6		0.8%	10	6986	88.7	75.5	82.1
	50	18353	82.1	85.8	78.3			12	3944	83.0	73.6	79.2
	55	10423	80.2	79.2	74.5			14	2367	82.1	70.8	67.9
RT2:QT-D7	55	39008	82.1	77.4	62.3		0.4%	12	7825	74.5	77.4	75.5
	60	20905	80.2	78.3	63.2			14	4486	72.6	76.4	77.4
	65	11998	80.2	77.4	69.8			16	2759	68.9	68.9	67.9
RT3	15	n/a					0.2%	15	1586	95.0	73.0	90.5
	20							20	840	94.0	63.5	92.5
	25							25	531	92.0	69.0	90.5

For the RT3 data, as demonstrated in Section 7.2, the standard FSM algorithms fail to discover interesting patterns when applied to RT3, due to the out-of-memory errors. Thus, only the classification accuracy results obtained using patterns discovered by gSpan-UBW is shown in Table 8.3. As can be seen from the table, using only a small number of features, the classifiers achieve quite good accuracy.

When using SW4, the performance of the classifiers built using patterns discovered by gSpan-UBW, as shown in Table B.14 exhibits the similar behaviour to the performance of the classifiers when using gSpan-UBW with SW1.

8.2.2 The evaluation of the UBW scheme on undirected graphs

Three groups of data, RG1, RG2 and RG3, that feature undirected graphs were used to evaluate the UBW scheme. As mentioned earlier, for the RG1 data, the structural weighting functions, SW1 and SW4, were used with gSpan-UBW; for the RG2 and RG3 data, the CW1-E function was used with gSpan-UBW. The experimental results with respect to each group of data are presented in the following paragraphs.

Efficiency test. The performance of gSpan-UBW on the RG1 data is shown in Table 8.4. In the table, the same range of support thresholds was used for the standard

FSM algorithms and gSpan-UBW respectively. As can be seen clearly from the table, using CH1, gSpan-UBW requires much less runtime and identifies a remarkably smaller number of patterns than gSpan while FFSM and GASTON can not operate using the support thresholds of below 16%. For the CH2 data, Table 8.4 shows that gSpan-UBW runs constantly slower than gSpan but runs faster than FFSM when the support threshold is over 6%, and GASTON can not operate using the support thresholds of below 10%. However, the number of patterns identified by gSpan-UBW on CH2 is significantly less than that discovered by the standard FSM algorithms. This fact may suggest that the benefit of using the UBW scheme on the CH2 data is at the expense of a prolonged runtime. It may also be inferred that the pruning strategy used in the UBW scheme is not effective when applied to CH2, resulting in a much longer runtime to filter the uninteresting patterns.

Table 8.4: The performance of gSpan-UBW with SW1 on the RG1 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-UBW + SW1			
		gSpan	FFSM	GASTON		μ	τ(%)	runtime	# patterns
RG1:CH1	10	1968.265	n/a	n/a	10995	0.2%	10	439.983	171
	12	1352.122	n/a	n/a	7009		12	297.321	120
	14	989.547	n/a	n/a	4909		14	181.210	83
	16	798.799	n/a	n/a	3596		16	113.912	63
RG1:CH2	4	572.614	740.596	n/a	8624	0.2%	4	1203.255	305
	6	335.605	506.909	n/a	3939		6	452.759	163
	8	243.612	289.341	n/a	2284		8	260.755	107
	10	171.225	242.826	n/a	1502		10	189.931	80

Additionally, when gSpan-UBW was coupled with the SW4 function, the performance of gSpan-UBW, as shown in Table 8.5, exhibits very similar behaviour to that of gSpan-UBW using SW1.

Table 8.5: The performance of gSpan-UBW using SW4 on the RG1 data

Dataset	μ	τ(%)	gSpan-UBW + SW4	
			runtime (in seconds)	# patterns
RG1:CH1	0.2%	10	444.757	164
		12	300.316	116
		14	183.833	79
		16	115.891	59
RG1:CH2	0.2%	4	1199.579	304
		6	454.725	162
		8	263.157	106
		10	191.070	79

When using the RG2 data, the performance of gSpan-UBW with CW1-E is presented in Table 8.6. As can be seen in the table, using the same range of support thresholds, gSpan runs faster than gSpan-UBW. When the support threshold is at 15%, the runtime of gSpan-UBW is very near to that of gSpan, but the difference between the number of patterns discovered by gSpan-UBW and the number of patterns discovered by gSpan is substantial. Thus, it may be inferred that the benefit of using

the UBW scheme is pronounced only when using low support thresholds (i.e. below 15%).

Table 8.6: The performance of gSpan-UBW with CW1-E on the RG2 and RG3 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-UBW + CW1-E			
		gSpan	FFSM	GASTON		μ	τ(%)	runtime	# patterns
RG2:MAM-V80	15	67.208	90.949	83.185	54621	0.02%	15	68.636	3240
	17	20.110	24.742	22.701	13044		17	67.922	3239
	19	9.381	10.310	8.418	3843		19	67.690	3208
	21	8.029	8.425	5.467	2958		21	61.054	2955
RG2:MAM-V100	15	167.893	247.678	198.546	95868	0.02%	15	174.905	2796
	17	53.441	65.258	45.597	22688		17	174.281	2796
	19	21.728	24.707	13.948	5991		19	169.240	2796
	21	19.719	21.064	11.282	4784		21	159.653	2796
RG3:BS-V500	8	4.992	3.588	8.716	24989	0.1%	8	14.498	505
	10	2.278	1.462	3.072	6285		10	7.350	437
	12	1.685	0.779	1.813	2073		12	4.579	407
	14	1.388	0.584	1.482	1229		14	3.367	377
RG3:BS-V1000	4	2.660	1.618	5.486	16833	0.04%	4	10.153	1079
	6	0.92	0.448	1.332	1404		6	3.641	744
	8	0.671	0.312	1.084	716		8	2.235	515
	10	0.421	0.246	0.958	443		10	1.521	369

In the case of the RG3 data, the performance of gSpan-UBW with CW1-E is also presented in Table 8.6. In the table, both the standard FSM algorithms and gSpan-UBW operate well but gSpan-UBW apparently runs slower than the standard FSM algorithms. In spite of this fact, gSpan-UBW identifies substantially fewer patterns than the standard FSM algorithms.

Table 8.7: The accuracy of the classifiers using patterns discovered by gSpan-UBW with SW1 on the CH1 data

Dataset	gSpan					vs.	gSpan-UBW					
	σ(%)	#F	Accuracy				μ	τ(%)	#F	Accuracy		
			NBC	SVM	C4.5					NBC	SVM	C4.5
RG1:CH1	16	3596	77.4	80.1	79.5		0.2%	10	171	73.9	76.5	76.9
	18	2735	76.8	79.6	79.2			12	120	74.5	76.6	78.2
	20	2149	76.8	79.5	80.2			14	83	71.7	75.0	76.5

Classification evaluation. Similar to the evaluations of other weighting schemes to the CH1 data, the AUC measure is also used in addition to the accuracy measure. Using the similar procedure described in Chapter 6, the accuracy results obtained using classifiers built with patterns discovered by gSpan-UBW with SW1 are presented in Table 8.7. As can be seen in the table, although the accuracy of the classifiers using patterns discovered by the standard FSM algorithms is moderately higher than that of the classifiers using patterns discovered by gSpan-UBW, the classifiers built using patterns discovered by gSpan-UBW use a remarkably smaller number of features.

The same phenomenon can also be observed in Table B.32 for the performance of gSpan-UBW with SW4. Further inspection of the AUC scores of the classifiers using patterns discovered by gSpan-UBW with SW1 or SW4, as shown in Table B.27,

indicates that using AUC scores, the performance of the classifiers built with patterns discovered by gSpan-UBW with SW1 or SW4 is far worse than that of the classifiers built with patterns discovered by the standard FSM algorithms. Thus, it may be inferred from these tables that the patterns discovered by gSpan-UBW with SW1 or SW4 are not as effective as those discovered by the standard FSM algorithms, despite the fact that gSpan-UBW with SW1 or SW4 discovers substantially fewer patterns than the standard FSM algorithms.

Table 8.8: The accuracy of the classifiers using patterns discovered by gSpan-UBW with CW1-E on the RG2 and RG3 data

Dataset	standard FSM algorithms					μ	gSpan-UBW + CW1-E				
	σ(%)	#F	NBC	SVM	C4.5		τ(%)	#F	NBC	SVM	C4.5
RG2:MAM-V80	18	6728	77.4	76.5	71.3	0.02%	15	3240	76.5	75.7	68.7
	20	3228	74.8	76.5	73.0		20	3128	76.5	75.7	68.7
	22	2698	76.5	75.7	68.7		25	1382	76.5	75.7	79.1
RG2:MAM-V100	18	11097	84.3	80.0	67.8	0.02%	10	3124	78.3	83.5	68.7
	20	5084	80.0	81.7	72.2		15	2796	80.0	81.7	65.2
	22	4538	80.9	80.9	69.6		20	2796	80.0	81.7	65.2
RG3:BS-V500	8	24989	97.6	95.9	83.5	0.1%	4	1297	97.1	93.5	86.5
	10	6285	96.5	93.5	84.7		6	620	97.1	94.1	87.1
	12	2073	93.5	92.9	86.5		8	505	95.3	92.9	81.8
RG3:BS-V1000	4	16833	95.9	92.4	84.7	0.08%	4	773	95.9	92.4	81.8
	6	1404	95.3	91.2	84.1		6	596	93.5	91.8	79.4
	8	716	92.9	91.2	84.1		8	479	94.7	91.8	80.6

In the case of RG2 and RG3, as described before, a two-class classification problem can be formulated for the respective RG2 and RG3 data. The performance of the classifiers built using patterns discovered by gSpan-UBW with CW1-E on the RG2 and RG3 data is shown in Table 8.8. In comparison with the performance of the classifiers built using patterns discovered by the standard FSM algorithms, Table 8.8 reveals that the classifiers built using patterns discovered by gSpan-UBW achieve very similar results to those built using patterns discovered by the standard FSM algorithms, but a considerably smaller number of features is required by the former than the latter.

8.2.3 The evaluation of the UBW scheme on directed graphs

The directed graphs employed in this thesis include the RG4 and RG5 data. As discussed in Chapter 6, only gSpan and gSpan-UBW were tested in the experiments. The experimental results obtained applying the UBW scheme to RG4 and RG5 are summarized in the following paragraphs.

Efficiency test. The RG4 data contains three data sets: IMDB, Amazon, and Ohsumed. The performance of gSpan-UBW, in comparison with gSpan, on the RG4 data is shown in Table 8.9. In the table, the same range of support thresholds was used respectively for gSpan and gSpan-UBW on IMDB, Amazon, and Ohsumed. As can be seen in the table, gSpan-UBW requires more runtime but discovers far fewer patterns than gSpan. Thus, the gain of reducing the number of patterns discovered by

gSpan-UBW on the RG4 data is achieved at the cost of more runtime. Is it worthwhile to identify a considerably smaller number of patterns but with increased runtime cost? The answer to this question is determined by the quality of the patterns discovered by gSpan-UBW. The evaluation of the quality of the patterns discovered by gSpan-UBW is considered below.

Table 8.9: The performance of gSpan-UBW with CW1-E on the RG4 and RG5 data

Dataset	gSpan			gSpan-UBW + CW1-E			
	σ(%)	runtime (in seconds)	# patterns	μ	τ(%)	runtime	# patterns
RG4:IMDB	0.1	6.738	8768	0.2%	0.1	38.329	5523
	0.2	4.888	3932		0.2	13.837	3320
	0.4	3.965	1866		0.4	4.883	1781
	0.6	3.126	1230		0.6	3.729	1204
RG4:Amazon	0.4	8.184	4680	0.08%	0.4	108.56	2880
	0.6	7.341	2901		0.6	45.818	2171
	0.8	7.051	1974		0.8	21.466	1658
	1.0	6.406	1534		1.0	14.617	1354
RG4:Ohsumed	0.4	7.391	4138	0.2%	0.4	62.790	2647
	0.6	5.762	2630		0.6	31.777	2011
	0.8	5.366	1902		0.8	19.344	1584
	1.0	5.178	1521		1.0	12.121	1319
RG5:Lancashire	10	5.691	8107	0.1%	10	3.339	1010
	12	2.891	3520		12	2.402	872
	14	2.010	1898		14	1.826	758
	16	1.575	1211		16	1.498	663
RG5:Scotland	10	43.836	84342	0.1%	10	6.380	3116
	12	11.341	20895		12	4.040	2612
	14	5.531	7856		14	3.073	2248
	16	3.685	3952		16	2.730	1912
RG5:GB	15	197.525	59081	0.02%	15	132.974	16428
	18	108.260	25248		18	87.189	13448
	20	79.076	17001		20	71.417	11750
	22	53.836	11540		22	54.990	9691

The RG5 social network data contains three data sets: Lancashire, Scotland and GB. The performance of gSpan-UBW with CW1-E on the RG5 data is also presented in Table 8.9. It can be clearly seen from the table that gSpan-UBW discovers considerably fewer patterns with significantly less runtime than gSpan, using the same range of support thresholds.

Classification evaluation. Since the graphs in RG4 have class labels, the quality of the patterns discovered by gSpan-UBW was evaluated using the framework of the frequent pattern based classification. The classification results are presented in Table 8.10. In comparison with the performance of the classifiers built using patterns discovered by gSpan, Table 8.10 suggests that the classifiers built using patterns discovered by gSpan-UBW retain the same level of accuracy but require much fewer features.

The graphs in the RG5 data do not feature any class labels and thus the quality of the patterns discovered cannot be assessed using a classification scenario.

Table 8.10: The accuracy of the classifiers using patterns discovered by gSpan-UBW with CW1-E on the RG4 data

Dataset	gSpan					vs.	μ	gSpan-UBW + CW1-E				
	σ(%)	#F	NBC	SVM	C4.5			τ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.1	8768	72.9	71.8	73.0		0.2%	0.2	3320	72.9	72.3	73.1
	0.2	3932	72.9	72.1	73.0			0.4	1781	72.9	72.6	73.1
	0.4	1866	72.9	72.5	73.1			0.6	1204	72.9	72.6	73.1
RG4:Amazon	0.4	4680	92.4	92.7	91.2		0.08%	0.4	2880	92.4	92.7	91.2
	0.6	2901	92.4	92.7	91.2			0.6	2171	92.4	92.7	91.2
	0.8	1974	92.5	93.0	91.2			0.8	1658	92.5	93.0	91.2
RG4:Ohsumed	0.4	4138	76.9	78.5	74.7		0.2%	0.4	2647	76.9	78.7	74.7
	0.6	2630	76.9	78.2	74.7			0.6	2011	76.9	78.4	74.7
	0.8	1902	76.9	77.8	74.7			0.8	1584	76.9	78.0	74.7

8.3 Summary and Discussion

An alternative subgraph weighting scheme that does not maintain the DCP was examined in this chapter. The evaluation of the UBW scheme was carried out by applying gSpan-UBW to various data sets. Since the implementation of the UBW scheme necessitates edge weightings, both the structural weighting and content weighting functions were employed. For data sets that do not have an edge weighting, the SW1 and SW4 weighting functions were assumed to be able to quantify the strength of the edges. However, for certain data sets, the SW1 and SW4 functions were found not to produce a good edge weighting, in terms of the effectiveness of the discovered patterns. For data sets that do have an edge weighting, the CW1-E function was used.

The performance of gSpan-UBW coupled with structural weighting functions differs, depending on the effectiveness of SW1 or SW4. For the RG1:CH1 data set, gSpan-UBW identifies substantially fewer patterns with significantly less runtime than gSpan. However the patterns discovered by gSpan-UBW were found to be less effective than those discovered by gSpan. For the RT1:CSLOGS-1 and RT1:CSLOGS-2 data sets, the advantage of gSpan-UBW over the standard FSM algorithms was prominent only when using low support thresholds. Further, the patterns discovered by the gSpan-UBW algorithm on the RT1:CSLOGS-1 and RT1:CSLOGS-2 data sets were found to be as effective as those discovered by the standard FSM algorithms, in terms of the classification accuracy. For the ST2, RT2, and RT3 data, the performance of gSpan-UBW surpassed that of the standard FSM algorithms with respect to both runtime and the number of patterns discovered. Moreover, the patterns discovered by gSpan-UBW on these data were as effective as those discovered by the standard FSM algorithms. For the ST1, RT1:CSLOGS-ALL, and RG1:CH2 data sets, gSpan-UBW discovered far fewer patterns than the standard FSM algorithms, despite the fact that the former ran considerably slower than the latter. However the quality of the patterns could not be assessed due to the unavailability of appropriate class labels.

The performance of gSpan-UBW varied with respect to data sets using the CW1-E

function. For the RG5 data set, gSpan-UBW outperformed gSpan, in terms of both runtime and the number of patterns discovered. For the RG2, RG3, and RG4 data sets, gSpan-UBW runs much slower than the standard FSM algorithms but identifies considerably fewer patterns than the standard FSM algorithms. Furthermore, the patterns discovered by gSpan-UBW with RG2 are slightly less effective than those discovered by the standard FSM algorithms; the patterns discovered by gSpan-UBW when applied to RG3 and RG4 were found to be as effective as those discovered by the standard FSM algorithms, in terms of their classification accuracy. It may be inferred from the performance of gSpan-UBW on RG2, RG3, and RG4 that the benefit of using the UBW scheme is usually at the expense of increasing the runtime, and that the pruning strategy used in the UBW scheme is not as strong as that achieved using subgraph weighting schemes that maintain the DCP. This is especially true with respect to data sets such as RG2, RG3 and RG4 where no high computation costs were incurred.

Altogether, the implementation of the UBW scheme is sensitive to the nature of the edge weightings, on some data sets, the operation of gSpan-UBW was found to be superior to that of the standard FSM algorithms, in terms of: (i) the runtime cost, (ii) the number of patterns identified, and (iii) the effectiveness of the patterns. On certain data sets, the benefits of the UBW scheme were achieved at the expense of increased runtime; and, in most cases, the effectiveness of the discovered patterns was retained.

Chapter 9

Further Discussions

In this chapter, based on the experimental results delineated in Chapters 6, 7 and 8, further discussions of different weighting functions and various subgraph weighting schemes are considered. The discussions are presented in Section 9.1 for weighting functions, and Section 9.2 for subgraph weighting schemes.

9.1 The Usage of Weighting Functions

According to Chapter 3, eight weighting functions were studied in this thesis. Except that the CW1 function is completely assigned by the domain user, the rest of the seven weighting functions utilize either the structure of the data or some partially provided domain knowledge of the data to compute the weights for vertexes or edges. Because the design of each of these seven weighting functions requires that each weighting function has to be used together with each suitable subgraph weighting scheme, the usage of each weighting function has to be considered in the context of each applicable subgraph weighting scheme.

Excluding CW1, as indicated before, the applicability of the seven weighting functions can be explained as follows. Specifically, SW1 and SW4 were applicable only to ATW, AW, and UBW; SW2, SW3, CW2, and CW3 were applicable only to CMW; SW5 was applicable only to ATW. In the context of the ATW scheme, SW1 and SW4 are edge weightings while SW5 is a vertex weighting; the performance of ATW coupled with SW1, SW4, or SW5 on the applicable data sets is very similar. In the context of the AW or UBW scheme, there is no big difference between the usage of the SW1 and SW4 functions with respect to the performance of AW or UBW on the applicable data sets. In the context of the CMW scheme, SW2 and SW3 are two edge weightings that are devised when the class labels are not available, while CW2 and CW3 are two edge weightings that are devised when the class labels are accessible; the performance of CMW coupled with SW2 is comparable to that of CMW coupled with SW3, and the same applies to the performance of CMW using CW2 and CW3.

Using SW1 is straightforward, because the computation of SW1 requires only the

size of the graph database (in terms of the total number of edges) and the number of occurrences of each edge in the database, which can be generated on the fly in the mining process. Although the computation of SW2 and SW3 is complex, it only involves computing the number of occurrences of each vertex and edge in the database, and such computing can usually be reduced by using some nice properties derived from SW2 and SW3. More importantly, SW2 and SW3 incorporate some feature selection measures, which are beneficial to extract statistically meaningful patterns, when the CMW scheme is coupled with SW2 or SW3. The computation of SW4 and SW5 considers not only the number of occurrences of the edges, but also the number of edges incident to the vertexes. Thus, SW4 and SW5 are more suitable to generate weights for vertexes or edges in graphs which have a lot of edge connections among vertexes. Similar to SW2 and SW3, CW2 and CW3 also incorporate feature selection measures but they require the domain knowledge. Using CW2 and CW3, in conjunction with the CMW scheme, is beneficial to extract discriminative patterns for the classification.

9.2 The Usage of Subgraph Weighting Schemes

Table 9.1: The applicability of subgraph weighting schemes with respect to various data sets

Data	Subgraph weighting scheme				
	ATW	AW	CMW	JSW	UBW
ST1	√	√	√	√	⊤
ST2	√	√	√	√	√
RT1:CSLOGS-ALL	√	×	√	√	⊤
RT1:CSLOGS-1	√	√	∗	√	⊤
RT1:CSLOGS-2	√	√	∗	√	⊤
RT2	√	×	×	√	√
RT3	×	√	√	√	√
RG1:CH1	√	×	√	√	×
RG1:CH2	√	√	⊤	√	⊤
RG2	√	×	√	√	⊤
RG3:BS-V500	√	√	∗	√	⊤
RG3:BS-V1000	√	×	⊤	√	⊤
RG4	√	×	√	√	⊤
RG5	√	√	√	√	√

Five different subgraph weighting schemes, using various strategies under different scenarios, were proposed in this thesis. The evaluation of these five subgraph weighting schemes on different data sets derived from different application domains suggests that except JSW, there is no other subgraph weighting scheme that is applicable to all the data sets. Table 9.1 indicates the applicability of all these five subgraph weighting schemes. With reference to the table, the '√', '×' , '∗', and '⊤' symbols in individual

cells indicate the following. The '$\surd$' symbol in a cell indicates that when using the corresponding subgraph weighting scheme on the corresponding data set, the weighted FSM algorithms run faster and discover considerably fewer patterns than standard FSM algorithms. The '$\times$' symbol indicates that the corresponding subgraph weighting scheme is not applicable to the corresponding data set. The '$*$' symbol indicates that when using the corresponding data set, the weighted FSM algorithms discover considerably fewer patterns than standard FSM algorithms but the runtime cost of the former is close to that of the latter. The '$\top$' symbol indicates that when using the corresponding data set, the weighted FSM algorithms discover considerably fewer patterns than standard FSM algorithms but the former requires more runtime than the latter.

As can be seen from the table, the UBW scheme is computationally the most expensive weighting scheme, in comparison with the other schemes, because the implementation of the UBW scheme violates the DCP while the implementations of the others do not. For the rest of the four weighting schemes, the JSW and ATW schemes appeared to be the two best performer; the AW scheme was the worst in terms of the applicability; the CMW scheme required more runtime than both ATW and JSW.

Table 9.1 further shows that some subgraph weighting schemes operate well on certain data and not so well on other data. Since each subgraph weighting scheme has its own weaknesses and strengths, and its advantages are only exemplified on certain data, it is not very useful to compare these five weighting schemes directly with respect to all the data sets used in the evaluation. However the research objective of this thesis is to demonstrate that weighted FSM algorithms can identify significantly fewer patterns with a reasonable amount of time compared to standard FSM algorithms and that the identified patterns are the "right" patterns. Thus the comparison of these weighting schemes are based on the number of patterns discovered and the effectiveness of the patterns, in terms of their classification effectiveness. Excluding the data that did not feature class labels, the data used for the classification experiments consisted of ST2, RT1:CSLOGS-1, RT1:CSLOGS-2, RT2, RT3, RG1:CH1, RG2, RG3, and RG4. Thus, the specific comparisons of different subgraph weighting schemes in terms of the number of patterns (i.e. features) identified and the effectiveness of the classifiers using such patterns are examined in the following paragraphs.

For the ST2 data, the weighted FSM algorithms using any of the five subgraph weighting schemes performed better than standard FSM algorithms, in terms of both the runtime cost and the number of patterns discovered. Table 9.2 compares the accuracy results obtained between the classifiers built using patterns discovered by weighted FSM algorithms and the classifiers built using patterns discovered by standard FSM algorithms. In the table, '$\#F_{nbc}$' , '$\#F_{c4.5}$' and '$\#F_{svm}$' denote the number of features used to construct the NBC, C4.5 and SVM classifiers respectively (these symbols will be used throughout the rest of this chapter unless otherwise specified). The best

Table 9.2: The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the ST2 data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
ST2:IM1000-D4	ATW	357	**93.6**	2173	94.9	357	95.7
	AW	297	92.5	273	95.1	189	**95.8**
	CMW	175	89.6	175	95.2	666	95.0
	JSW	823	90.6	2001	**95.5**	823	95.7
	UBW	270	93.5	179	94.4	179	95.7
	No Weighting	7263	92.9	7263	94.9	1659	95.5
ST2:IM1000-D5	ATW	1550	86.2	1829	91.6	1501	91.6
	AW	2000	86.2	1188	91.3	1188	91.6
	CMW	1514	86.9	1514	91.3	1682	88.9
	JSW	2155	84.1	794	91.1	2155	90.2
	UBW	1567	**90.3**	1567	**92.3**	1567	**92.9**
	No Weighting	6287	87.0	6287	91.9	1163	91.4
ST2:IM1000-D6	ATW	2126	81.2	1462	86.7	1462	75.2
	AW	2011	81.2	1130	86.7	396	77.1
	CMW	490	**90.9**	4901	87.5	4901	75.9
	JSW	3526	81.3	1634	89.6	3526	78.4
	UBW	1106	85.2	1106	**89.8**	1106	**79.3**
	No Weighting	6936	81.9	6936	81.9	6936	76.4

accuracy for each classifier under each category is highlighted in bold. As shown in Table 9.2, the patterns discovered by the weighted FSM algorithms were found to be at least as effective as those discovered using standard FSM algorithms, in terms of the classification performance. In general, with respect to the number of features used to build the three classifiers, UBW appears to be the best, and is followed by AW and CMW; ATW and JSW employ the largest number of features.

Table 9.3: The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the CSLOGS-1(2) data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
RT1:CSLOGS-1	ATW	286	79.8	286	79.8	391	81.1
	AW	439	80.5	718	**82.3**	300	81.7
	CMW	629	80.7	695	82.2	629	81.8
	JSW	177	**80.7**	242	81.8	417	**81.8**
	UBW	303	80.6	303	82.1	645	81.8
	No Weighting	1286	79.8	2582	81.8	2582	81.8
RT1:CSLOGS-2	ATW	292	80.4	292	81.5	383	81.7
	AW	685	80.6	685	82.3	685	81.9
	CMW	678	80.4	678	**82.6**	678	81.5
	JSW	186	**80.6**	429	82.3	417	81.8
	UBW	314	80.6	314	82.2	314	**82.4**
	No Weighting	1281	80.4	2485	82.1	2485	82.0

With respect to the RT1:CSLOGS-1 and RT1:CSLOGS-2 data sets, the performance of both standard FSM and weighted FSM algorithms was very similar. Using the RT1:CSLOGS-1 data set as an example, the weighted FSM algorithms using any of the ATW, AW, and JSW schemes run faster and identified significantly fewer patterns than standard FSM algorithms. The weighted FSM algorithms coupled with either of

the CMW and UBW schemes, required more computation effort to discover considerably fewer patterns than the standard FSM algorithms, especially when the support threshold was relatively high. As for the quality of the patterns discovered by the weighted FSM algorithms, Table 9.3 lists the accuracy results of the classifiers built using patterns discovered by the weighted FSM algorithms integrating with the five weighting schemes. As can be seen from the table, the patterns discovered using any of the AW, CMW, JSW and UBW schemes were found to be slightly more effective than those discovered using standard FSM algorithms; the JSW scheme seemed to be the best among these four proposed schemes in terms of classification accuracy and number of features used to build the classifiers. Table 9.3 also indicates that the patterns discovered using weighted FSM algorithms coupled with the ATW scheme appeared to be marginally less effective than those discovered using standard FSM algorithms. However the classifiers built using the ATW patterns require significantly fewer features than those built using patterns discovered by the standard FSM algorithms.

Table 9.4: The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RT2 data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
RT2:QT-D5	ATW	2186	73.6	4877	67	2186	66.0
	JSW	18820	**88.7**	18820	74.5	18820	72.6
	UBW	4450	75.5	2561	**80.2**	2591	70.8
	No Weighting	36955	79.2	36955	68.9	36955	**76.4**
RT2:QT-D6	ATW	17736	82.1	17736	78.3	18065	**85.8**
	JSW	21591	88.7	21591	**88.7**	21591	76.4
	UBW	6986	**88.7**	6713	82.1	6713	79.2
	No Weighting	41096	85.8	18353	78.3	18353	85.8
RT2:QT-D7	ATW	2790	**85.8**	2790	**78.3**	4918	74.5
	JSW	18022	84.0	18022	72.6	20730	77.4
	UBW	7420	74.5	4486	77.4	4175	**78.3**
	No Weighting	39008	82.1	6959	70.8	20905	78.3

For the RT2 data, weighted FSM algorithms coupled with any of the ATW, JSW, and UBW schemes outperformed standard FSM algorithms with respect to both the runtime cost and the number of patterns found. In addition, the classifiers built using pattens discovered by the weighted FSM algorithms achieve a very similar performance to those built using patterns discovered by the standard FSM algorithms. As can be seen from Table 9.4, among the three proposed weighting schemes, using the JSW scheme required substantially more features, than the other two schemes, to build the classifiers and using the UBW scheme seemed to obtain the best result, in terms of classification accuracy and the number of features used to build the classifiers. The AW and CMW schemes were not applicable to the RT2 data because of the reasons explained in Sub-section 7.1.1.

For the RT3 data, standard FSM algorithms simply did not work. Among the five subgraph weighting schemes, the weighted FSM algorithms coupled with the ATW

scheme failed to find effective patterns due to the out-of-memory errors. The weighted FSM algorithms using any of the rest of the four weighting schemes worked very well on the RT3 data. As can be seen from Table 9.5, in comparison with the CMW, JSW and UBW schemes, the best classification results were achieved (with the smallest number of features) using the AW scheme.

Table 9.5: The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RT3 data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
RT3	AW	1600	98.5	942	**94.0**	942	**88.5**
	CMW	3390	96.0	339	91.5	2442	80.5
	JSW	2702	**99.0**	1532	93.0	2702	87.0
	UBW	1539	95.0	812	92.5	1539	73.0
	No Weighting	n/a	n/a	n/a	n/a	n/a	n/a

For the RG1:CH1 data, the weighted FSM algorithms coupled with any of the ATW, CMW, and JSW schemes required significantly less runtime and found considerably fewer patterns than standard FSM algorithms. For these three weighting schemes, Table 9.6 shows that the classifiers built using patterns discovered by the weighted FSM algorithms achieved comparable performance with those built using patterns discovered by the standard FSM algorithms, but the former needed a significantly smaller number of features than the latter. Table 9.6 further shows that in the light of the classifiers' performance and the number of features required to build the classifiers, both the CMW and JSW schemes appeared to be better than the ATW scheme. Additionally, using the AW and UBW weighting scheme, the weighted FSM algorithms failed to identify effective patterns with respect to classification accuracy, although it ran faster and discovered far fewer patterns than the standard FSM algorithms.

Table 9.6: The AUC scores of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG1:CH1 data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
RG1:CH1	ATW	1656	79.1	2382	76.2	1474	70.9
	CMW	1516	75.2	399	76.2	491	**71.9**
	JSW	404	73.3	404	**76.6**	404	69.1
	No Weighting	3596	**79.5**	2149	76.0	3596	69.3

For the RG2 data, the standard FSM algorithms worked well and the advantages of the weighted FSM algorithms over the standard FSM algorithms were not prominent when using relatively high support thresholds. The weighted FSM algorithms coupled with any of the ATW, CMW and JSW schemes performed better than the standard FSM algorithms with respect to the runtime cost and the number of patterns discovered. Further, the AW scheme did not work on the RG2 data, and the weighted FSM algorithms using the UBW scheme required more runtime than the standard FSM algorithms, in order to identify fewer patterns. As can be seen from Table 9.7, in terms of

the classification accuracy using patterns discovered by the weighted FSM algorithms, using the CMW scheme was the most cost effective choice, and accuracy results obtained using the remaining three schemes were very similar, but using the UBW scheme required considerably fewer features than the ATW scheme, to build the classifiers.

Table 9.7: The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG2 data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
RG2:MAM-V80	ATW	2126	77.4	1809	73.0	2126	77.4
	CMW	1500	**86.1**	623	76.5	353	**78.3**
	JSW	1276	77.4	2226	73.0	2226	76.5
	UBW	1382	76.5	1382	**79.1**	1382	75.7
	No Weighting	6728	77.4	3228	73.0	3228	76.5
RG2:MAM-V100	ATW	5476	83.5	5316	71.3	5476	84.3
	CMW	6962	**94.8**	928	**72.2**	3809	**86.1**
	JSW	2495	80.0	3903	70.4	2495	81.7
	UBW	2796	80.0	3124	68.7	3124	83.5
	No Weighting	11097	84.3	5084	72.2	5084	81.7

Table 9.8: The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG3 data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
RG3:BS-V500	ATW	6162	97.6	1341	87.6	1341	94.7
	AW	1171	96.5	1171	83.5	4923	**95.9**
	CMW	1153	97.6	776	82.9	2674	95.3
	JSW	10858	**98.2**	10858	**88.2**	1858	95.3
	UBW	620	97.1	620	87.1	620	94.1
	No Weighting	24989	97.6	2073	86.5	24989	95.9
RG3:BS-V1000	ATW	911	95.9	1071	84.1	766	**92.9**
	CMW	2413	95.3	400	**85.3**	2413	92.9
	JSW	10262	**96.5**	715	84.7	10262	91.8
	UBW	773	95.9	773	81.8	773	92.4
	No Weighting	16833	95.9	443	85.3	16833	92.4

For the RG3 data, the standard FSM algorithms worked well. For the RG3:BS-V500 data, the weighted FSM algorithms using any of the ATW, AW, CMW and JSW weighting schemes required less runtime to discover significantly fewer patterns than the standard FSM algorithms. The weighted FSM algorithm using the UBW scheme was found to run slower than the standard FSM algorithms, but the former discovered substantially fewer patterns than the latter. As can be seen from Table 9.8, using the UBW scheme, the classifiers achieved a similar accuracy result to those when using other schemes, but used a much smaller number of features; using the JSW scheme, the classifiers had a tendency to employ a much larger number of features than using other schemes, to achieve a similar level of accuracy. As for the RG3:BS-V1000 data, the AW scheme did not work; and the weighted FSM algorithms using either the CMW or UBW scheme required more runtime cost, incurred by the computation required to generate the weightings, than that using the ATW or JSW scheme. In comparison

with the accuracy of the classifiers built using patterns discovered by the standard FSM algorithms, the accuracy of the classifiers built using patterns discovered by the weighted FSM algorithms with any of the ATW, CMW, JSW and UBW schemes, as shown in Table 9.8, was found to be at a comparable level.

Table 9.9: The accuracy of the classifiers using patterns discovered by weighted FSM algorithms with respect to various weighting schemes on the RG4 data

Data	Weighting scheme	$\#F_{nbc}$	NBC (%)	$\#F_{c4.5}$	C4.5 (%)	$\#F_{svm}$	SVM (%)
RG4:IMDB	ATW	1203	72.9	1203	73.1	1203	72.6
	CMW	1203	72.9	1203	73.1	1203	72.6
	JSW	1203	72.9	1203	73.1	1203	72.6
	UBW	1204	72.9	1204	73.1	1204	72.6
	No Weighting	1866	72.9	1866	73.1	1866	72.5
RG4:Amazon	ATW	1653	92.5	1653	91.2	1653	93.0
	CMW	1662	92.5	1662	91.2	1662	93.0
	JSW	1653	92.5	1653	91.2	1653	93.0
	UBW	1658	92.5	1658	91.2	1658	93.0
	No Weighting	1974	92.5	1974	91.2	1974	93.0
RG4:Ohsumed	ATW	1582	**77.4**	1582	**75.4**	2646	78.7
	CMW	1592	76.9	1592	74.7	2674	78.7
	JSW	1583	77.4	1583	75.4	2671	**79.3**
	UBW	1584	76.9	1584	74.7	2647	78.7
	No Weighting	1902	76.9	1902	74.7	4138	78.5

For the RG4 data, the standard FSM algorithms worked well, whilst the AW scheme failed to work. The weighted FSM algorithms using the ATW, CMW and JSW schemes mostly outperformed the standard FSM algorithms, in terms of the runtime cost and the number of patterns discovered. The UBW scheme required more runtime than the other workable weighting schemes. As can be seen from Table 9.9, the classifiers built using patterns discovered by the weighted FSM algorithms outperformed those built using patterns discovered by the standard FSM algorithms in terms of the classification accuracy and number of features used to build the classifiers. Using any of the four weighting schemes (ATW, CMW, JSW, and UBW), the performance of classifiers built using patterns discovered by the weighted FSM algorithms were found to be very similar.

By and large, there is no constant "winner" among the five subgraph weighting schemes. Practically, the effectiveness of the subgraph weighting schemes was found to be related to the following "factors": (i) the characteristics of the specific graph representation, (ii) the specific weighting function adopted, and (iii) the purpose of the application. An appropriate graph representation necessitated a good knowledge of the application domain and a proper weighting function relied on both the graph representation and domain knowledge of the application. The performance of the weighted FSM algorithms using any of the weighting schemes may be significantly affected by an unsuitable graph representation or an unfitting weighting function.

Precisely, the UBW scheme was computationally the most expensive because it did

not satisfy the DCP, resulting in a much longer time to filter uninteresting patterns. However, the UBW scheme mostly tends to identify a relatively smaller number of patterns than the ATW and JSW schemes. Further, applying the UBW scheme can utilize the domain knowledge represented by edge weightings, and the patterns identified by using UBW are effective, in terms of the performance of the classifiers.

For the remaining of four weighting schemes, which all satisfied the DCP, the JSW scheme did not use any weighting functions for vertexes or edges and the performance of the weighted FSM algorithm using the JSW scheme was mainly affected by the graph structure used to calculate the weight for each discovered subgraph. Thus, the computational complexity of employing JSW, as indicated in Table 9.1, is not high. In addition, no domain knowledge is required for applying JSW. The CMW scheme mostly required more runtime than the ATW, AW, and JSW schemes, because the computational overhead required by the SW2, SW3, CW2, and CW3 weighting functions is time consuming, especially for large data sets. Nevertheless, the CMW scheme has the capacity to incorporate the domain knowledge represented by using feature selection measures, which can be utilized to extract meaningful patterns for the classification. Similar to UBW, the AW scheme was found to be very sensitive to the specific weighting function, because an unfitting edge weighting might lead to the ineffectiveness of the pruning strategy used in AW. Compared with UBW, the use of AW is more efficient than that of UBW, because computing the weighting ratio in the AW scheme is rather simple and more importantly, the weighting ratio satisfies the DCP. On the contrary to both AW and UBW, the ATW scheme was found to be not sensitive to the specific weighting functions used. Therefore, similar to JSW, applying the ATW scheme is straightforward and does not require the domain knowledge. Further, as demonstrated in Chapters 6 and 7, the application of the ATW scheme is suitable for the graph representation using hierarchical decomposition (e.g. ST2 and RT2).

Overall, how to choose the appropriate subgraph weighting scheme can be suggested as follows. When no domain knowledge is available, ATW, JSW, CMW coupled with SW2 or SW3, and UBW coupled with SW1 or SW4 are preferred; when the user provides the class labels for the data, CMW coupled with CW2 or CW3 is preferred; when the edge weightings provided by the user are accurate, ATW, AW and UBW are preferred. Alternatively, if the computational complexity is of main concern, ATW and JSW are used firstly, and followed by AW and CMW; UBW is used as the final resort.

Chapter 10

Conclusion

This chapter presents the main findings, the research contribution and possible future directions. The main findings of the research are presented in Section 10.1, and the contribution of the research work and directions for future research in Section 10.2 and Section 10.3 respectively.

10.1 Findings

In this thesis, the concept of weight was introduced to assign "importance" to each discovered subgraph so that a framework of weighted FSM could be formulated to reduce the computational complexity incurred by FSM, but still retain the quality of the discovered patterns. The weighted FSM framework included three components: (i) graph representations, (ii) weighting functions for vertexes or edges, and (iii) subgraph weighting schemes. These three components were intertwined in the design of the weighted FSM algorithms. The proposed weighted FSM framework was designed to address the research question introduced in Chapter 1.2, and to answer each sub-question related to this research question. A detailed analysis per sub-question is presented in the following.

(a) **What form should the desired weighting function take?** The weighting functions introduced in Chapter 4 can be employed to assign weights for either vertexes or edges. However, as discussed in Chapters 6 and 8, the AW, CMW and UBW schemes all required an edge weighting, the ATW scheme required either a vertex or edge weighting, and the JSW scheme did not need any vertex or edge weights. Thus, the selection of the specific weighting function was dependent on the usage of the specific subgraph weighting scheme. In the case of the ATW scheme, both the edge weighting (i.e. SW1, SW4) and vertex weighting (i.e. SW5) had similar effects on the performance of the weighted FSM algorithms using the ATW scheme.

(b) **How should weightings be derived?** Again, the answer to this question is similar to question (a). Whether to employ user provided weights or structural weighting was application dependent. In practice, it is usually not easy to obtain user provided weights, especially for large data collections. Under this situation, using alternative structural weighting is a necessity.

(c) **What is the nature of the data structures that would be required to support weighted frequent subgraph mining?** As illustrated in Chapter 3, different types of graphs (trees, undirected graphs and directed graphs) were constructed from different domains of application. An ability to operate with different types of graphs is therefore essential for weighted frequent subgraph mining. According to the experimental analysis described in Chapters 6, 7, and 8, the weighted FSM algorithms using the five proposed subgraph weighting schemes were able to support trees, undirected graphs and directed graphs irrespective of the differences of their structures.

(d) **How should weightings be used?** As discussed in Section 9.2, the effectiveness of the subgraph weighting scheme relied on the characteristics of the graph representation, the weighting function used, and the application goal. Thus, effective subgraph weighting necessitates taking into consideration both the graph representation and the weighting function with respect to the application objective.

(e) **Whether to maintain the Downward Closure Property (DCP) or not?** As indicated by Table 9.1, the UBW scheme that did not maintain the DCP was found to be computationally less efficient than the other four weighting schemes that maintained the DCP. Therefore, it can be concluded that it is desirable to devise subgraph weighting schemes that maintain the DCP, in terms of computational efficiency.

Altogether, the research conducted in this thesis revealed the following facts.

(a) Weighted FSM, regardless of the weighting scheme used, identifies significantly smaller number of patterns than the standard FSM algorithms.

(b) Weighted FSM using weighting schemes that maintain the DCP generally runs significantly faster than the standard FSM algorithms.

(c) In most cases, weighted FSM using the UBW scheme requires more runtime than the standard FSM algorithms.

(d) The patterns discovered using weighted FSM, coupled with any of the five weighting schemes, are at least as effective as those discovered by the standard FSM algorithms with respect to the classification power (i.e. the "correct" frequent subgraphs and subtrees are discovered).

10.2 Contributions

With respect to Section 1.5, the primary contributions of the research work presented in this thesis can be listed as follows:

(a) The *weighting* concept, which puts an emphasis on the most important frequent subgraphs instead of identifying all the frequent subgraphs during the mining process.

(b) The derivation of a number of weighting functions to determine the weights (significance) of vertexes or edges in graphs. As described in Chapter 4, five structural weighting functions: SW1, SW2, SW3, SW4 and SW5 were proposed to compute automatically the weights for vertexes or edges in graphs, and three content weighting functions: CW1, CW2 and CW3 were proposed to compute the weights for vertexes or edges in graphs by taking the domain knowledge into consideration.

(c) A sequence of subgraph weighting schemes, to attach significance to identified subgraphs, which can be integrated seamlessly into the process of mining frequent subgraphs. As described in Chapters 5 and 8, four subgraph weighting schemes that maintained the DCP and one alternative subgraph weighting scheme that did not maintain the DCP were proposed.

(d) A number of weighted FSM algorithms founded on different weighting strategies and directed at different types of graphs. As described in Chapters 6 and 8, these were integrated into a number of standard FSM algorithms (gSpan, FFSM and GASTON) so that they could be evaluated.

(e) A framework for integrating *feature selection* techniques into the weighted frequent subgraph mining process in the context of frequent pattern based graph classification. The application of the CMW scheme proposed in Chapter 5 required the SW2, SW3, CW2 and CW3 weighting functions discussed in Chapter 4. All these four weighting functions employed one of the feature selection techniques: (i) Phi correlation coefficient, (ii) normalized mutual information, and (iii) the χ^2 measure. The evaluation of the weighted FSM algorithms using the CMW scheme, as discussed in Chapter 6, indicated that the patters discovered by the weighted FSM algorithms using the CMW scheme coupled with CW2 or CW3 could be directly fed into the classifiers to gain a good classification accuracy without applying feature selection.

(f) A mechanism for the domain user to control the mining process by adjusting either the support or the weight threshold or both, such that the computational complexity of frequent subgraph mining is reduced in a trade-off between efficiency and effectiveness. Except that the ATW scheme used a weighted support threshold to

control the mining result, the other four weighting schemes all required two parameters: (i) (weighted) support threshold, and (ii) weighting threshold, to control the mining result.

(g) A new framework of image classification using an image interest points based graph representation and weighted frequent subgraph mining. In Chapter 3, both the RG2 (Mammography) and RG3 (photographic image) data were modelled as graphs using an image interest points based representation. The performance of the classifiers built using patterns discovered by the weighted FSM algorithms on either RG2 or RG3, as shown in Chapters 6 and 8, indicated that by using an image interest points based graph representation, in conjunction with weighted FSM, classifiers built using patterns discovered by weighted FSM can achieve good classification accuracy.

(h) A systematic framework for classifying documents described using a graph based representation, in conjunction with the weighted frequent subgraph mining. In Chapter 3, the RT3 data was modelled as trees using a semantic graph based representation, and the RG4 data was modelled as directed graphs using a term occurrence based representation. As illustrated in Chapters 6, 7 and 8, the performance of the classifiers built using patterns discovered by the weighted FSM algorithms on both RT3 and RG4 suggested that weighted FSM algorithms combined with the graph based representation can facilitate classifying documents on a large scale.

10.3 Future Directions

The research described in this thesis has sparked a number of promising directions for future research.

- **Dynamic social networks mining**. Graphs created from social network data tend to change dynamically with time. Using graph mining approaches to identify interesting patterns in dynamic social networks has become a very active area of research. Some examples include analysing the properties of time-evolving graphs [Leskovec et al., 2005], mining dynamic frequent subgraphs [Berger-Wolf and Saia, 2006]. The weights for vertexes or edges can also change with time in dynamic social networks. Thus, how to work with dynamic weights with respect to social network mining presents an interesting problem.

- **Graph based image representations using image interest points.** The idea of representing images as a vocabulary of individual visual words, of which, each visual word is represented by a cluster of image interest points, has already proved its merit for image classification [Lazebnik et al., 2006, Jiang et al.,

2007]. However, how best to choose the visual words properly, with regard to the computation issue, is still unsettled. For the image representations described in Sub-sections 3.2.5 and 3.2.6, images were modelled as graphs where each vertex represents the visual word, and each edge the relationship between any two visual words. In this representation, each vertex was weighted by the number of interest points contained in the visual word represented by that vertex, and each edge was weighted by some similarity measure. Thus, using this representation, the problem of choosing visual words is actually transformed into a weighted frequent subgraph mining problem. Each discovered weighted frequent subgraph can contain one or more visual words connected according to some similarity value. The future challenge with respect to such image representations is to create more compact graph representations for images with more effective vertex or edge weighting, so that weighted FSM algorithms can be used to extract more discriminative patterns, which lead to good image classification accuracy.

- **Graph based document representation.** Although using graph based representations for documents is not a new idea, few researchers have applied weighted FSM algorithms to graph represented documents on a large scale basis. The research studies using semantic or keyword based graph representations for documents (as described in Sub-sections 3.2.3 and 3.2.7) can be further extended to model large collections of documents (e.g. Reuters 21578, 20 newsgroups). More importantly, how to label the vertexes or edges, and assign meaningful weights to them is necessary if we wish to achieve effective graph based text mining.

- **Integrating feature selection techniques into the FSM algorithms.** Feature selection plays an important role in the framework for frequent pattern based classification. Is it possible to incorporate feature selection techniques into weighted FSM so as to directly identify the most discriminative weighted subgraphs which are effective with respect to the classification task? There is still much room for researchers to utilize classic data mining techniques and integrate them into weighted FSM.

- **Parallel weighted frequent subgraph mining.** The inherent combinatorial complexity of the frequent subgraph mining process continues to present a challenge as we wish to mine graph sets of ever increasing size. The weighted frequent subgraph mining techniques proposed in this thesis can alleviate this computation complexity to some degree. However, when the size of the data to be mined is very large (for example the ST1:T1M data introduced in Sub-section 3.1.1 contains 1000000 trees), the computational overhead becomes unacceptable. In this case, using distributed/parallel weighted frequent subgraph mining may provide a solution. One piece of research work directed at this theme is reported in Fatta

and Berthold [2005] where an extension of the MoFa algorithm (described in Sub-section 2.6.1.2) is presented to accommodate the distributed computation of mining frequent subgraphs with respect to data sets representing large molecular compounds.

Bibliography

A. Abello, M.G.C. Resende, and S. Sundarsky. Massive quasi-clique detection. In *Proceedings of the 5th Latin America Symposium on Theoretical Informatics*, pages 598–612, 2002.

M. Aery and S. Chakravarthy. InfoSift: Adapting graph mining techniques for text classification. In *Proceedings of the 8th International Florida Artificial Intelligence Research Society Conference*, 2005a.

M. Aery and S. Chakravarthy. eMailSift: Email classification based on structure and content. In *Proceedings of the 5th IEEE International Conference on Data Mining*, 2005b.

R.C. Agarwal, C.C. Aggarwal, and V.V.V. Prasad. A tree projection algorithm for generation of frequent itemsets. *Journal of Parallel and f Computing*, 61(3):350–371, 2001.

R. Agrawal and R. Srikant. Fast algorithm for mining association rules. In *Proceedings of the 20th International Conference on Very Large Databases (VLDB'94)*, pages 487–499. Morgan Kaufmann, 1994.

R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases. In *Proceedings of 1993 ACM SIGMOD International Conference on Management of Data*, pages 207–216, 1993.

R. Albert, H. Jeong, and A.L. Barabási. Diameter of the World-Wide Web. *Nature*, 401:130–131, 1999.

E. Alm and A.P. Arkin. Biological networks. *Current Opinion in Structural Biology*, 13(2):193–202, 2003.

T. Asai, K. Abe ans S. Kawasoe, H. Arimura, H. Sakamoto, and S. Arikawa. Efficient substructure discovery from large semi-structured data. In *Proceedings of the 2nd SIAM International Conference on Data Mining (SDM'02)*, pages 158–174, 2002.

T. Asai, H. Arimura, T. Uno, and S. Nakano. Discovering frequent substructures in large unordered trees. In *Proceedings of the 6th International Conference on Discovery Science*, pages 47–61, 2003.

D. Avis and K. Fukuda. Reverse search for enumeration. *Discrete Applied Mathematics*, 65(1-3):21–46, 1996.

B. Barber and H.J. Hamilton. Extracting share frequent itemsets with infrequent subsets. *Journal of Data Mining and Knowledge Discovery*, 7:153–185, 2003.

E. Barbu, P. Héroux, S. Adam, and E. Trupin. Clustering document image using graph summaries. In *Proceedings of the 5th International Conference on Learning and Data Mining*, pages 194–202, 2005.

A. Barrat, M. Barthélémy, R. Pastor-Satorras, and A. Vespignani. The architecture of complex weighted networks. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 101(11), pages 3747–3752, 2004.

R.J. Bayardo Jr. Efficiently mining long patterns from databases. In *Proceedings of the 1998 International Conference on Management of Data (SIGMOD'98)*, pages 85–93, 1998.

T. Y. Berger-Wolf and J. Saia. A framework for analysis of dynamic social networks. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 523–528, 2006.

Biology-Online.org. Biology-online dictionary. `http://www.biology-online.org/dictionary/Phylogeny`, June 2010.

I.M. Bomze, M. Budinich, P.M. Pardalos, and M. Pelillo. The maximum clique problem. In *Handbook of Combinatorial Optimization*, pages 1–74. Kluwer Academic Publishers, 1999.

F. Bonchi, F. Giannotti, A. Mazzanti, and D. Pedreschi. Exante: Anticipated data reduction in constrained pattern mining. In *Proceedings of the 7th European Conference on Principles and Practice of Knowledge Discovery in Databases (PKDD)*, 2003.

C. Borgelt and M. Berthold. Mining molecular fragments: finding relevant substructures of molecules. In *Proceedings of the 2002 International Conference on Data Mining*, pages 211–218, 2002.

K.M. Borgwardt and H.P. Kriegel. Shortest-path kernels on graphs. In *Proceedings of the 2005 International Conference on Data Mining*, pages 74–81, 2005.

K.M. Borgwardt, H.P. Kriegel, and P. Wackersreuther. Pattern mining in frequent dynamic subgraphs. In *Proceedings of the 2006 IEEE International Conference on Data Mining*, pages 818–822, 2006.

S. Brin and L. Page. The anatomy of a large scale hyper-textual web search engine. In *Proceedings of the 7th International World Wide Web Conference*, 1998.

B. Bringmann and A. Zimmermann. One in a million: picking the right patterns. *Knowledge and Information Systems*, 18(1):62–81, 2009.

H. Bunke and G. Allerman. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1(4):245–253, 1983.

H. Bunke, P. Foggia, C. Guidobaldi, C. Sansone, and M. Vento. A comparison of algorithms for maximum common subgraph on randomly connected graphs. In *Proceedings of the Joint IAPR International Workshop on Structural,Syntactic, and Statistical Pattern Recognition*, pages 123–132, 2002.

C.H. Cai, A.W. Fu, C.H. Cheng, and W.W. Kwong. Mining association rules with weighted items. In *Proceedings of International Database Engineering and Applications Symposium*, 1998.

D. Cai, Z. Shao, X. He, X. Yan, and J. Han. Mining hidden community in heterogeneous social networks. In *Proceedings of ACM SIGKDD Workshop on Link Discovery:Issues and Approaches and Applications*, 2005.

T. Calders, J. Ramon, and D. van Dyck. Anti-monotonic overlap-graph support measures. In *Proceedings of the Eighth IEEE International Conference on Data Mining*, pages 73–82, 2008.

C.L. Carter, H.J. Hamilton, and N. Cercone. Share based measure for itemsets. In *Proceedings of the 1st European Conference on the Principles of Data Mining and Knowledge Discovery*, volume 1263, pages 14–24, 1997.

D. Chakrabarti and C. Faloutsos. Graph mining: laws, generators, and algorithms. *ACM Computing Surveys*, 38(1), 2006.

S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Mining the link structure of the world wide web. *IEEE Computer*, 32(8):60–67, 1999.

C. Chang and C. Lin. *LIBSVM: A library for support vector machines*, 2001. Software available at `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

G. Chartrand and P. Zhang. *Introduction to Graph Theory (1st Edition)*. McGraw Hill Science/Engineering/Math, 2004.

C. Chen, X. Yan, P.S. Yu, J. Han, D. Zhang, and X. Gu. Towards graph containment search and indexing. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*, pages 926–937, 2007a.

C. Chen, X. Yan, F. Zhu, and J. Han. gApprox: Mining frequent approximate patterns from a massive network. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 445–450, 2007b.

C. Chen, C.X. Lin, X. Yan, and J. Han. On effective presentation of graph patterns: a structural representative approach. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 299–308, 2008.

M.S. Chen, J. Han, and Philip S. Y. Data mining: an overview from database perspective. *IEEE Transactions on Knowledge and Data Engineering*, 8:866–883, 1996.

Y. Chi, Y. Yang, Y. Xia, and R.R. Muntz. Indexing and mining free trees. In *Proceedings of the 2003 IEEE International Conference on Data Mining (ICDM'03)*, 2003a.

Y. Chi, Y. Yang, Y. Xia, and R.R. Muntz. CMTreeMiner: Mining both closed and maximal frequent subtrees. Technical Report CSD-TR No. 030053, University of California, Los Angeles, 2003b.

Y. Chi, R.R. Muntz, S. Nijssen, and J.N. Kok. Frequent subtree mining - an overview. *Fundamenta Informaticae*, 66(1-2):161–198, 2004a.

Y. Chi, Y. Yang, Y. Xia, and R.R. Muntz. CMTreeMiner: Mining both closed and maximal frequent subtrees. In *Proceedings of the 8th Pacific Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, 2004b.

Y. Chi, Y. Yang, Y. Xia, and R.R. Muntz. 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 (SSDBM'04)*, page 11, 2004c.

Y. Chi, Y. Yang, Y. Xia, and R.R. Muntz. Canonical forms for labelled trees and their applications in frequent subtree mining. *Journal of Knowledge and Information Systems*, 8(2):203–234, 2005.

W.J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):749–764, 1995.

M.J. Chung. $\mathcal{O}(n^{2.5})$ time algorithm for the subgraph homomorphism problem on trees. *Journal of Algorithms*, 13:106–112, 1987.

K.W. Church and P. Hanks. Word association norms, mutual information and lexicography. *Computational Linguistics*, 16:22–29, 1990.

A. Clauset, M.E.J. Newman, and C. Moore. Finding community in very large networks. *Physical Review E*, 70:066111, 2004.

Frans Coenen. The LUCS-KDD Random Image Generator. `http://www.csc.liv.ac.uk/~frans/KDD/Software/ImageGenerator/imageGenerator.html`, January 2009.

D. Conte, F. Foggia, C. Sansone, and M. Vento. Thirty years of graph matching in pattern recognition. *International Journal of Pattern Recognition and Artificial Intelligence*, 18(3):265–298, 2004.

D. Conte, P. Foggia, and M. Vento. Challenging complexity of maximum common subgraph detection algorithms: A performance analysis of three algorithms on a wide database of graphs. *Journal of Graph Algorithms and Applications*, 11(1):99–143, 2007.

D.J. Cook and L.B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, 1994.

D.J. Cook and L.B. Holder. Graph based data mining. *IEEE Intelligence Systems*, 15 (2):32–41, 2000.

R. Cooley, B. Mobasher, and J. Srivastava. Web Mining: information and pattern discovery on the World Wide Web. In *Proceedings of the 9th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'97)*, 1997.

L.P. Cordella, P. Foggia, C. Sansone, F. Tortorella, and M. Vento. Graph matching: a fast algorithm and its evaluation. In *Proceedings of the 14th Conference on Pattern Recognition (ICPR'98)*, pages 1582–1584, 1998.

L.P. Cordella, P. Foggia, C. Sansone, and M. Vento. An improved algorithm for matching large graphs. In *Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition*, pages 149–159, 2001.

T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms (2nd Edition)*. MIT Press and McGraw-Hill, 2001.

T.M. Cover and J.A. Thomas. *Elements of Information Theory*. John Wiley & Sons, Incorporation, 1991.

J. Cui, J. Kim, D. Maggiorini, K. Boussetta, and M. Gerla. Aggregated multicast- a comparative study. *Cluster Computing*, 8(1):15–26, 2005.

Luc Dehaspe, Hannu Toivonen, and Ross D. King. Finding frequent substructures in chemical compounds. In *Proceedings of the Fourth International Conference on Knowledge Discovery and Data Mining (KDD'98)*, pages 30–36, New York, August 1998. AAI Press.

M. Deshpande, M. Kuramochi, and G. Karypis. Frequent substructure based approaches for classifying chemical compounds. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1036–1050, 2005.

P. Desikn and J. Srivastava. Mining temporally evolving graphs. In *Proceedings of the WebKDD2004*, pages 22–25, 2004.

J. Diesner, T.L. Frantz, and K.M. Carley. Communication networks from the enron email corpus "It's always about the people. Enron is no different". *Computational & Mathematical Organization Theory*, 11(3):201–228, 2005.

C. Ding, X. He, H. Zha, M. Gu, and H. Simon. A min-max cut algorithm for graph partitioning and data clustering. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, 2001.

H. Ebel, L.I. Mielsch, and S. Bornholdt. Scale-free topology of e-mail networks. *Physical Review E*, 66(3):035103, 2002.

W. Eberle and L.B. Holder. Discovering structural anomalies in graph based data. In *Proceedings of the 7th IEEE International Conference on Data Mining Workshops*, pages 393–398, 2007.

F. Eichinger, K. Böhm, and M. Huber. Mining edge-weighted call graphs to localise software bugs. In *Proceedings of the 8th European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2008.

A. Elsayed, F. Coenen, C. Jiang, M. Garcia-Finana, and V. Sluming. Corpus callosum MR image classification. *Knowledge Based Systems*, 23(4):330–336, 2010.

M. Faloutsos, P. Faloutsos, and C. Faloutsos. On power-law relationships of the internet topology. In *Proceedings of the ACM Special Interest Group on Data Communications (SIGCOMM'99)*, pages 251–262. ACM Press, 1999.

W. Fan, K. Zhang, H. Cheng, J. Gao, X. Yan, J. Han, P.S. Yu, and O. Verscheure. Direct mining of discriminative and essential graphical and itemset features via model-based search tree. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2008.

G.D. Fatta and M.R. Berthold. High performance subgraph mining in molecular compounds. In *Proceedings of the 2005 International Conference on High Performance Computing and Communications (HPCC'05)*, pages 866–877, 2005.

L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: an incremental bayesian approach tested on 101 object categories. In *IEEE CVPR2004, Workshop on Generative-Model Based Vision*, 2004.

R.A. Finkel and J.L. Bentley. Quad-trees, a data structure for retrieval on composite keys. *Acta Informatica*, 4(1):1–9, 1974.

I. Fischer and T. Meinl. Graph based molecular data mining - an overview. In *Proceedings of the 2004 IEEE International Conference on Systems,Man and Cybernetics*, pages 4578–4582, 2004.

G. Flake, S. Lawrence, and C.L. Giles. Efficient identification of web communities. In *Proceedings of the Conference of the ACM Special Interest Group on Knowledge Discovery and Data Mining*. ACM Press, 2000.

G. Flake, R. Tarjan, and K. Tsioutsiouliklis. Graph clustering and minimum cut trees. *Internet Mathematics*, 1:385–408, 2004.

P. Foggia, R. Genna, and M. Vento. A performance comparison of five algorithms for graph isomorphism. In *Proceedings of the 3rd IAPR-TC15 Workshop on Graph-based Representation in Pattern Recognition*, pages 188–199, 2001.

S. Fortin. The graph isomorphism problem. Technical Report TR96-20, The University of Alberta, 1996.

L. Freeman. Centrality in social networks conceptual clarification. *Social Networks*, 1 (3):215–239, 1979.

X. Gao, B. Xiao, and D. Tao X. Li. A survey of graph edit distance. *Pattern Analysis & Applications*, 13(1):113–129, 2010.

M. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New York, 1979.

T. Gärtner, P. Flach, and S. Wrobel. On graph kernels: hardness results and efficient alternatives. In *Proceedings of the Sixteenth Annual Conference on Learning Theory (COLT'03)*, pages 129–143, 2003.

L. Getoor and C. Diehl. Link mining: A survey. *ACM SIGKDD Explorations Newsletter*, 7(2):3–12, 2005.

A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

M. Girvan and M.E.J. Newman. Community structure in social and biological networks. In *Proceedings of the National Academy of Science of the United States of America*, volume 99(12), pages 7821–7826, 2002.

G. Greco, A. Guzzo, G. Manco, and D. Saccà. Mining and reasoning on work-flows. *IEEE Transactions on Knowledge and Data Engineering*, 17(4):519–534, 2005.

P.D. Grünwald. *The Minimum Description Length Principle*. The MIT Press, 2007.

S. Gudes, S.E. Shimony, and N. Vanetik. Discovering frequent graph patterns using disjoint paths. *IEEE Transaction on Knowledge and Data Engineering*, 18(11):1441–1456, 2006.

G. Gutin. 5.3 independent sets and cliques. In J.L. Gross and J. Yellin, editors, *Handbook of Graph Theory, Discrete Mathematics & Its Applications*, pages 389–402. CRC Press, 2004.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I.H. Witten. The WEKA data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

J. Han and M. Kamber. *Data Mining Concepts and Techniques (2nd Edition)*. San Francisco: Morgan Kaufmann, 2006.

J. Han, J. Pei, and Y. Yin. Mining frequent patterns without candidate generation. In *Proceedings of ACM SIGMOD International Conference on Management of Data*, pages 1–12. ACM Press, 2000.

J. Han, H. Cheng, D. Xin, and X. Yan. Frequent pattern mining: current status and future directions. *Data Mining and Knowledge Discovery*, 15(1):55–86, 2007.

F. Harary and I.C. Ross. A procedure for clique detection using the group matrix. *Sociometry*, 20(3):205–215, 1957.

M.A. Hasan, V. Chaoji, S. Salem, J. Besson, and M.J. Zaki. ORIGAMI: Mining representative orthogonal graph patterns. In *Proceedings of the 7th IEEE International Conference on Data Mining*, pages 153–162, 2007.

J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. *Discrete Applied Mathematics*, 71:153–169, 1996.

S. Hido and H. Kawano. 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, 2005.

J.E. Hopcroft and R.E. Tarjan. Isomorphism of planar graphs. In R.E. Miller and J.W. Thatcher, editors, *Complexity of Computer Computations*, pages 131–152. Plenum Press, 1972.

H. Hu, X. Yan, Y. Huang, J. Han, and X.J. Zhou. Mining coherent dense subgraphs across massive biological networks for functional discovery. *Bioinformatics*, 21(1): 213–221, 2005.

J. Huan, W. Wang, and J. Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In *Proceedings of the 2003 International Conference on Data Mining (ICDM'03)*, pages 549–552, 2003.

J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining spatial motifs from protein structure graphs. In *Proceedings of the Eighth Annual International Conference on Research in Computational Molecular Biology (RECOMB'04)*, 2004a.

J. Huan, W. Wang, J. Prins, and J. Yang. SPIN: Mining maximal frequent subgraphs from graph databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 581–586, 2004b.

J. Huan, W. Wang, A. Washington, J. Prins, R. Shah, and A. Tropsha. Accurate classification of protein structural families based on coherent subgraph analysis. In *Proceedings of Pacific Symposium on Biocomputing*, pages 411–422, 2004c.

X. Huang and W. Lai. Clustering graphs for visualization via node similarities. *Visual Languages and Computing*, 17:225–253, 2006.

G. Ifrim and G. Weikum. Transductive learning for text classification using explicit knowledge models. In *Proceedings of the Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 223–234. Springer Lecture Notes in Artificial Intelligence, 2006.

G. Ifrim, G. Bakir, and G. Weikum. Fast logistic regression for text categorization with variable-length n-grams. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 354–362, 2008.

A. Inokuchi, T. Washio, and H. Motoda. An apriori-based algorithm for mining frequent substructure from graph data. In *Proceedings of the Fourth European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 13–23, 2000.

A. Inokuchi, T. Washio, K. Nishimura, and H. Motoda. A fast algorithm for mining frequent connected subgraphs. Research Report RT0448, IBM Research, Tokyo Research Laboratory, 2002.

A. Inokuchi, T. Washio, and H. Motoda. Complete mining of frequent patterns from graphs: mining graph data. *Journal of Machine Learning*, 50(3):321–354, 2003.

C. Jiang and F. Coenen. Graph-based image classification by weighting scheme. In *Proceedings of the 2008 SGAI International Conference on Artificial Intelligence (AI'08)*, pages 63–76, 2008.

C. Jiang, F. Coenen, R. Sanderson, and M. Zito. Text classification using graph mining based feature extraction. In *Proceedings of the 2009 SGAI International Conference on Artificial Intelligence (AI'09)*, pages 21–34, 2009.

C. Jiang, F. Coenen, R. Sanderson, and M. Zito. Text classification using graph mining-based feature extraction. *Knowledge Based Systems*, 23(4):302–308, 2010a.

C. Jiang, F. Coenen, and M. Zito. Finding frequent subgraphs in longitudinal social network data using a weighted graph mining approach. In *Proceedings of the 6th International Conference on Advanced Data Mining and Applications (ADMA'10)*, pages 405–416. Springer LNCS, 2010b.

C. Jiang, F. Coenen, and M. Zito. Frequent sub-graph mining on edge weighted graphs. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery(DaWak'10)*, pages 77–88. Springer LNCS, 2010c.

Y.G. Jiang, C.W. Ngo, and J. Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *Proceedings of ACM International Conference on Image and Video Retrieval*, 2007.

N. Jin, C. Young, and W. Wang. Graph classification based on pattern co-occurrence. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 573–582, 2009.

W. Jin and R.K. Srihari. Graph-based text representation and knowledge discovery. In *Proceedings of the 2007 ACM Symposium on Applied Computing*, pages 807–811, 2007.

H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labelled graphs. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML'03)*, pages 321–328, 2003.

Y. Ke, J. Cheng, and W. Ng. Correlation search in graph databases. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'07)*, pages 390–399, 2007.

Y. Ke, J. Cheng, and J. Yu. Efficient discovery of frequent correlated subgraph pairs. In *Proceedings of the 9th IEEE International Conference on Data Mining*, pages 239–248, 2009.

Y.P. Ke, J. Cheng, and W. NG. Correlated pattern mining in quantitative databases. *ACM Transactions on Database Systems*, 33(3):14:1–14:45, 2008.

B. Kelley, R. Sharan, R. Karp, E. Sittler, D. Root, B. Stockwell, and T. Ideker. Conserved pathways within bacteria and yeast as revealed by global protein network

alignment. In *Proceedings of the National Academy of Sciences of the United States of America (PNAS'03)*, 2003.

J.M. Kleinberg. Authoritative sources in a hyperlinked environment. In *Proceedings of ACM SIAM Symposium Discrete Algorithms*, pages 668–677, 1998.

R. Kosala and H. Blockeel. Web mining research: A survey. *ACM SIGKDD Explorations Newsletter*, 2(1):1–15, 2000.

G. Kossinets and D.J. Watts. Empirical analysis of an evolving social network. *Science*, 311:88–90, 2006.

M. Koyutürk and W. Szpankowski. An efficient algorithm for detecting frequent subgraphs in biological networks. *Journal of Bioinformatics*, 20(supplement 1):200–207, 2004.

S. Kramer, L.D. Raedt, and C. Helma. Molecular feature mining in HIV data. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pages 136–143, 2001.

T. Kudo, E. Maeda, and Y. Matsumoto. An application of boosting to graph classification. In *Proceedings of Eighteenth Annual Conference on Neural Information Processing Systems (NIPS04)*, 2004.

M. Kuramochi and G. Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 International Conference on Data Mining*, pages 313–320, 2001.

M. Kuramochi and G. Karypis. Discovering frequent geometric subgraphs. In *Proceedings of the Second IEEE International Conference on Data Mining*, pages 258–, 2002.

M. Kuramochi and G. Karypis. GREW - a scalable frequent subgraph discovery algorithm. In *Proceedings of the Fourth IEEE International Conference on Data Mining*, pages 439–442, 2004a.

M. Kuramochi and G. Karypis. An efficient algorithm for discovering frequent subgraphs. *IEEE Transactions on Knowledge and Data Engineering*, 16(9):1038–1051, 2004b.

M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. In *Proceedings of the SIAM International Conference on Data Mining*, 2004c.

M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph. *Data Mining and Knowledge Discovery*, 11(3):243–271, 2005.

M. Lahiri and T.Y. Berger-Wolf. Structure prediction in temporal networks using frequent subgraphs. In *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Data Mining (CIDM'07)*, pages 35–42, 2007.

S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: spatial pyramid matching for recognizing natural scene categories. In *Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, pages 2169–2178, 2006.

J. Leskovec, J. Kleinberg, and C. Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 177–187, 2005.

X. Li, B. Liu, and P.S. Yu. Mining community structure of named entities from web pages and blogs. In *AAAI Symposium - Computational Approaches to Analysing Weblogs*, 2006.

B. Liu. *Web Data Mining: Exploring Hyper-links, Contents, and Usage Data.* Springer, 2008.

B. Liu, W. Hsu, and Y. Ma. Integrating classification and association rule mining. In *Proceedings of the 4th International Conference on Knowledge Discovery and Data Mining (KDD'98)*, 1998.

C. Liu, X. Yan, H. Yu, J. Han, and P.S. Yu. Mining behaviour graphs for 'backtrace' of non-crashing bugs. In *Proceedings of 2005 SIAM International Conference on Data Mining (SDM'05)*, pages 286–297, 2005.

T. Liu and D. Geiger. Approximate tree matching and shape similarity. In *Proceedings of the 1999 International Conference on Computer Vision*, 1999.

D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

S.W. Lu, Y. Ren, and C.Y. Suen. Hierarchical attributed graph representation and recognition of handwritten chinese characters. *Pattern Recognition*, 24:617–632, 1991.

A. Markov and M. Last. Efficient graph-based representation of web documents. In *Proceedings of the Third International Workshop on Mining Graphs, Trees and Sequences*, pages 52–62, 2005.

D.W. Matula. Subtree isomorphism in $\mathcal{O}(n^{5/2})$. *Annals of Discrete Mathematics*, 2: 91–106, 1978.

M. McGlohon, L. Akoglu, and C. Faloutsos. Weighted graphs and disconnected components. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 524–532, 2008.

J. McGregor. Backtrack search algorithms and the maximal common subgraph problem. *Software: Practice and Experience*, 12:13–23, 1982.

B.D. McKay. Practical graph isomorphism. *Congress Numerantium*, 30:45–87, 1981.

MedlinePlus. Bethesda (MD): U.S. National Library of Medicine. `http://www.nlm.nih.gov/medlineplus/ency/article/003791.htm`, October 2010.

B.T. Messmer. *Efficient Graph Matching Algorithms for Preprocessed Model Graphs.* PhD thesis, Institute of Computer Science and Applied Mathematics, University of Bern, 1996.

B.T. Messmer and H. Bunke. A new algorithm for error-tolerant subgraph isomorphism detection. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 20(5): 493–504, 1998.

K. Mikolajczyk and C. Schmid. An affine invariant interest point detector. In *Proceedings of the 7th European Conference on Computer Vision-Part I*, pages 128–142, 2002.

R. Milo, S. Shen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: simple building blocks of complex networks. *Science*, 298(5594):824–827, 2002.

T. Mitchell. *Machine Learning.* McGraw Hill, 1996.

T. Miyazaki. The complexity of mckay's canonical labeling algorithm. In *Groups and Computation II, DIMACS Series Discrete Mathematics Theoretical Computer Science*, volume 28, pages 239–256. American Mathematical Society, 1997.

M. Mukherjee and L.B. Holder. Graph-based data mining on social networks. In *Proceedings of the ACM KDD Workshop on Link Analysis and Group Detection*, 2004.

S. Nakano. Efficient generation of plane trees. *Information Processing Letters*, 84: 167–172, 2002.

A.A. Nanavati, S. Gurumurthy, G. Das, D. Chakraborty, K. Dasgupta, S. Mukherjea, and A. Joshi. On the structure properties of massive telecom call graphs: findings and implications. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 435–444, 2006.

M.E.J. Newman. Scientific collaboration networks: II shortest paths, weighted networks and centrality. *Physical Review E*, 64(016132):5835–5838, 2001.

M.E.J. Newman. Mixing patterns in networks. *Physical Review E*, 67(2):026126, 2003.

M.E.J. Newman. Fast algorithms for detecting community structure in networks. *Physical Review E*, 69(6):066133, 2004a.

M.E.J. Newman. Detecting community structure in networks. *The European Physical Journal B - Condensed Matter and Complex Systems*, 38(2):321–330, 2004b.

M.E.J. Newman. Analysis of weighted networks. *Physical Review E*, 70(056131), 2004c.

M.E.J. Newman and M. Girvan. Finding and evaluating community structure in networks. *Physical Review E*, 69:026113, 2004.

R. Ng, L.V.S. Lakshmanan, J. Han, and A. Pang. Exploratory mining and pruning optimizations of constrained associations rules. In *Proceedings of the 1998 ACM SIGMOD International Conference on Management of Data*, pages 13–24, 1998.

S. Nijssen and J. N. Kok. The gaston tool for frequent subgraph mining. *Electronic Notes in Theorectical Computer Science*, 127:77–87, 2005.

S. Nijssen and J.N. Kok. Efficient discovery of frequent unordered trees. In *Proceedings of the 1st International Workshop on Mining Graphs, Trees and Sequences (MGTS'03)*, 2003.

S. Nijssen and J.N. Kok. A Quickstart in frequent structure mining can make a difference. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 647–652, 2004.

C.C. Noble and D.J. Cook. Graph based anomaly detection. In *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 631–636, 2003.

D.L. Nowell and J. Kleinberg. The link prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007.

S. Nowozin, K. Tsuda, T. Uno, T. Kudo, and G. Bakir. Weighted substructure mining for image analysis. In *Proceedings of the 2007 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (ICCV'07)*, 2007.

T. Ozaki and T. Ohkawa. Mining correlated subgraphs in graph databases. In *Proceedings of the 12th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'08)*, pages 272–283, 2008.

S. Paul. *Multi-casting On the Internet and Its Applications.* Kluwer Academic Publishers, 1998.

A. Pearce, T. Caelli, and W.F. Bischof. Rule-graphs for graph matching in pattern recognition. *Pattern Recognition*, 27(9):1231–1246, 1994.

J. Pei, J. Han, and L.V.S. Lakshmanan. Mining frequent itemsets with convertible constraints. In *Proceedings of the 17th International Conference on Data Engineering*, pages 433–442, 2001a.

J. Pei, J. Han, B. Mortazavi-asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *Proceedings of the 12th IEEE International Conference on Data Engineering (ICDE'01)*, pages 215–224, 2001b.

J. Pei, D. Jiang, and A. Zhang. On mining cross-graph quasi-cliques. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2005.

B.R. Preiss. *Data Structures and Algorithms with Object-Oriented Design Patterns in C++.* Wiley, 1998.

F. Provost and T. Fawcett. Robust classification for imprecise environments. *Machine Learning*, 42(3):203–231, 2001.

J. Punin, M. Krishnamoorthy, and M. Zaki. LOGML: Log markup language for web usage mining. In *WEBKDD Workshop*, 2001.

J. Quinlan. *C4.5: Programs for Machine Learning.* Morgan Kaufmann Publishers, San Francisco, 1993.

L. Ralaivola, S.J. Swamidass, H. Saigo, and P. Baldi. Graph kernels for chemical informatics. *Neural Networks*, 18(8):1093–1110, 2005.

R.C. Read and D.G. Corneil. The graph isomorph disease. *Journal of Graph Theory*, 1:339–363, 1977.

S. Redner. How popular is your paper? an empirical study of the citation distribution. *The European Physical Journal B*, 4:131–134, 1998.

H.T. Reynolds. *The Analysis of Cross-Classifications.* The Free Press, New York, 1977.

S.E. Robinson and R.M. Christley. Identifying temporal variation in reported births, deaths and movements of cattle in britain. *Journal of BMC Veterinary Research*, 2 (11):10.1186/1746–6148–2–11, 2006.

U. Ruckert and S. Kramer. Frequent free tree discovery in graph data. In *Proceedings of the 2004 ACM Symposium on Applied Computing*, pages 564–570, 2004.

H. Saigo, N. Krämer, and K. Tsuda. Partial least squares regression for graph mining. In *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 578–586, 2008.

G. Salton, A. Wong, and C.S. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

A. Sanfeliu and K.S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Transactions on Systems,Man,and Cybernetics*, 13(3): 353–363, 1983.

R.J. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches*. John Wiley & Sons, Incorporation, 1992.

A. Schenker. *Graph Theoretic Techniques for Web Content Mining*. PhD thesis, University of South Forida, 2003.

A. Schenker, H. Bunke, M. Last, and A. Kandel. A graph-based framework for web document mining. In *Proceedings of the 6th International Workshop on Document Analysis Systems*, pages 401–412, 2004.

D.C. Schmidt and L.E. Druffel. A fast backtracking algorithm to test directed graphs for isomorphism using distance matrices. *Journal of the ACM*, 23(3):433–445, 1976.

B. Schölkopf and A.J. Smola. *Learning with Kernels*. The MIT Press, 2002.

F. Schreiber and H. Schwöbbermeyer. Frequency concepts and pattern detection for the analysis of motifs in networks. *Transactions on Computational Systems Biology*, 3:89–104, 2005.

R. Shamir and D. Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2): 267–280, 1999.

L.G. Shapiro and R.M. Haralick. Structural descriptions and inexact matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:504–519, 1981.

R. Sharan, S. Suthram, R. Kelley, T. Kuhn, S. McCuine, P. Uetz, T. Sittler, R. Karp, and T. Ideker. Conserved patterns of protein interaction in multiple species. In *Proceedings of the National Academy of Sciences of the United States of America (PNAS'05)*, 2005.

D. Shasha, J.T.L. Wang, and R. Giugno. Algorithms and applications of tree and graph searching. In *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART Symposium on Principles on Database Systems (PODS'02)*, pages 39–52, 2002.

D. Shasha, J.T.L. Wang, and S. Zhang. Unordered tree mining with applications to phylogeny. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, pages 708–719, 2004.

D.J. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. CRC Press, Boca Raton, FL, 1997.

J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 2000.

T.A.B. Snijders. The statistical evaluation of social network dynamics. *Sociological Methodology*, 31:361–395, 2001.

J. Suckling, J. Parker, D. Dance, S. Astley, I. Hutt, C. Boggis, I. Ricketts, E. Stamatakis, N. Cerneaz, S. Kok, P. Taylor, D. Betal, and J. Savage. The mammographic image analysis society digital mammogram database. *Excerpta Medica, International Congress Series*, 1069:375–378, 1994.

H. Tan, T. Dillon, F. Hadzic, E. Chang, and L. Feng. MB3-Miner: Mining embedded subtrees using tree model guided candidate generation. In *Proceedings of the 1st International Workshop on Mining Complex Data 2005 (MCD'05)*, pages 103–110, 2005a.

H. Tan, T.S. Dillon, L. Feng, E. Chang, and F. Hadzic. X3-Miner: Mining patterns from xml database. In *Proceedings of the 6th International Data Mining*, pages 287–297, Skiathos, Greece, 2005b.

H. Tan, T.S. Dillon, F. Hadzic, E. Chang, and L. Feng. IMB3-Miner: Mining induced/embedded subtrees by constraining the level of embedding. In *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'06)*, pages 450–461, 2006.

P.N. Tan, V. Kumar, and J. Srivastava. Selecting the right interestingness measure for association patterns. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 32–41, 2002.

F. Tao, F. Murtagh, and M. Farid. Weighted association rule mining using weighted support and significance framework. In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2003.

S. Tatikonda, S. Parthasarathy, and T. Kurc. Trips and Tides: New algorithm for tree mining. In *Proceedings of the 15th ACM International Conference on Information and Knowledge Management*, pages 455–464, 2006.

A. Termier, M.C. Rousset, and M. Sebag. Treefinder: a first step towards xml data mining. In *Proceedings of the 2002 IEEE International Conference on Data Mining (ICDM'02)*, pages 450–457, 2002.

M. Thoma, H. Cheng, A. Gretton, J. Han, H. Kriegel, A. Smol, L. Song, P. Yu, X. Yan, and K. Borgwardt. Near-optimal supervised feature selection among frequent subgraphs. In *Proceedings of the 2009 SIAM International Conference on Data Mining*, pages 1075–1086, 2009.

L.T. Thomas, S.R. Valluri, and K. Karlapalem. MARGIN: Maximal frequent subgraph mining. In *Proceedings of the 6th International Conference on Data Mining (ICDM'06)*, pages 1097–1101, 2006.

W. Tsai and K.S. Fu. Subgraph error-correcting isomorphism for synthetic pattern recognition. *IEEE Transactions on Systems,Man,and Cybernetics*, 13(1):48–62, 1983.

W.H. Tsai and K.S. Fu. Error correcting isomorphism of attributed relational graphs for pattern analysis. *IEEE Transactions on System, Man, and Cybernetics*, 9(12): 757–768, 1979.

K. Tsuda and T. Kudo. Clustering graphs by weighted substructure mining. In *Proceedings of the 23rd International Conference on Machine Learning (ICML'06)*, pages 953–960, 2006.

Y. Tsuruoka and J. Tsujii. Bidirectional inference with the easiest-first strategy for tagging sequence data. In *Proceedings of the Conference of Human Language Technology and Empirical Methods in Natural Language Processing*, pages 467–474, 2005.

T. Tuytelaars and K. Mikolajczyk. Local invariant feature detectors: a survey. *Foundations and Trends in Computer Graphics and Vision*, 3(3):177–280, 2008.

P. Uetz. Protein-protein interaction modelling. `http://www.visualcomplexity.com/vc/index.cfm?domain=Biology`, 2003.

J.R. Ullmann. An algorithm for subgraph isomorphism. *Journal of the ACM*, 23(1): 31–42, 1976.

G. Valiente. *Algorithms on Trees and Graphs*. Springer, 2002.

N. Vanetik. Discovery of frequent patterns in semi-structured data. Master's thesis, Department of Computer Science, Ben Gurion University, 2002.

N. Vanetik, E. Gudes, and S.E. Shimony. Computing frequent graph patterns from semi-structured data. In *Proceedings of the 2nd International Conference on Data Mining (ICDM'02)*, pages 458–465, 2002.

N. Vanetik, S.E. Shimony, and E. Gudes. Support measures for graph data. *Data Mining and Knowledge Discovery*, 13(2):243–260, 2006.

V. Vapnik. *Statistical Learning Theory.* Wiley InterScience, 1998.

V.N. vapnik. An overview of statistical learning theory. *IEEE Transactions on Neural Networks*, 10:988–999, 1999.

N.X. Vinh, J. Epps, and J. Bailey. Information theoretic measures for clusterings comparison: is a correction for chance necessary? In *Proceedings of the 26th International Conference on Machine Learning*, pages 1073–1080, 2009.

N. Wale and G. Karypis. Acyclic subgraph-based descriptor spaces for chemical compound retrieval and classification. In *Proceedings of the 2006 IEEE International Conference on Data Mining*, 2006.

C. Wang, M. Hong, J. Pei, H. Zhou, W. Wang, and B. Shi. Efficient pattern-growth methods for frequent tree pattern mining. In *Proceedings of the 8th Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'04)*, pages 441–451, 2004a.

C. Wang, W. Wang, J. Pei, Y. Zhu, and B. Shi. Scalable mining of large disk-based graph databases. In *Proceedings of the 10th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 316–325, 2004b.

J. Wang and J. Han. Bide: Efficient mining of frequent closed sequences. In *Proceedings of the 20th International Conference on Data Engineering*, pages 79–90, 2004.

J. Wang, Z. Zeng, and L. Zhou. Clan: An algorithm for mining closed cliques from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 797–802, 2006.

W. Wang, J. Yang, and P.S. Yu. Efficient mining of weighted association rule (war). In *Proceedings of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2000.

T. Washio and H. Motoda. State of the art of graph-based data mining. *SIGKDD Explorations*, 5:59–68, 2003.

S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications.* Cambridge University Press, New York, NY, 1994.

D.J. Watts and S.H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

O. Weislow, R. Kiser, D. Fine, J. Bader, R. Shoemaker, and M. Boyd. New soluble formazan assay for hiv-11 cytopathic effects: application to high flux screening of synthetic and natural products for aids antiviral activity. *Journal of the National Cancer Institute*, 81:577–586, 1989.

D. B. West. *Introduction to Graph Theory(2nd Edition)*. Prentice Hall, 2000.

S. White and P. Smyth. Algorithms for estimating relative importance in networks. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 266–275, 2003.

E.K. Wong. Model matching in robot vision by subgraph isomorphism. *Pattern Recognition*, 25(3):287–304, 1992.

M. Wörlein, T. Meinl, I. Fischer, and M. Philippsen. A quantitative comparison of the subgraph miners mofa, gspan, ffsm and gaston. In *Proceedings of the 9th European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 392–404, 2005.

F. Wu and B.A. Huberman. Finding communities in linear time: a physics approach. *The European Physical Journal B*, 38(2):331–338, 2004.

Y. Xiao, J.F. Yao, Z. Li, and M.H. Dunham. Efficient data mining for maximal frequent subtrees. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM'03)*, 2003.

D. Xin, H. Cheng, and X. Yan. Extracting redundancy aware top k patterns. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 444–453, 2006a.

D. Xin, H. Cheng, X. Yan, and J. Han. Extracting redundancy aware top k patterns. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 444–453, 2006b.

D. Xin, X. Shen, Q. Mei, and J. Han. Discovering interesting patterns through user's interactive feedback. In *Proceedings of the 13th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'06)*, pages 773–778, 2006c.

H. Xiong, S. Shekhar, P.N. Tan, and V. Kumar. Exploring a support-based upper bound of pearson's correlation coefficient for efficiently identifying strongly correlated pairs. In *Proceedings of the ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pages 334–343, 2004.

H. Xiong, P.N. Tan, and V. Kumar. Hyper-clique pattern discovery. *Journal of Data Mining and Knowledge Discovery*, 13(2):219–242, 2006.

Y. Xu, G. Jones, J.T. Li, B. Wang, and C.M. Sun. A study on mutual information-based feature selection for text categorization. *Journal of Computational Information Systems*, 1(2):203–213, 2005.

X. Yan and J. Han. Close graph: Mining closed frequent graph patterns. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 286–295, 2003.

X. Yan and J.W. Han. gspan: Graph-based substructure pattern mining. In *Proceedings of the 2002 International Conference on Data Mining*, page 721, 2002.

X. Yan, P.S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data (SIGMOD'04)*, pages 335–346, 2004.

X. Yan, P.S. Yu, and J. Han. Graph indexing based on discriminative frequent structure analysis. *ACM Transaction on Database Systems*, 30(4):960–993, 2005a.

X. Yan, P.S. Yu, and J. Han. Sub-structure similarity search in graph databases. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (SIGMOD'05)*, pages 766–777, 2005b.

X. Yan, X. Zhou, and J. Han. Mining closed relational graphs with connectivity constraints. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*, pages 324–333, 2005c.

X. Yan, F. Zhu, J. Han, and P.S. Yu. Searching substructures with superimposed distance. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, page 88, 2006a.

X. Yan, F. Zhu, P.S. Yu, and J. Han. Feature-based similarity search in graph structures. *ACM Transactions on Database Systems*, 31(4):1418–1453, 2006b.

X. Yan, M.R. Mehan, Y. Huang, M.S. Waterman, P.S. Yu, and X.J. Zhou. A graph-based approach to systematically reconstruct human transcriptional regulatory modules. In *Proceedings of 15th Annual International Conference on Intelligent Systems for Molecular Biology (ISMB)*, 2007.

X. Yan, H. Cheng, J. Han, and Philip S. Yu. Mining significant graph patterns by leap search. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 433–444, 2008.

J. Yang, Y.G. Jiang, A.G. Hauptmann, and C.W. Ngo. Evaluating bag-of-visual-words representations in scene classification. In *Proceedings of the International Workshop on Multimedia Information Retrieval*, pages 197–206, 2007.

L.H. Yang, M.L. Lee, W. Hsu, and S. Achary. Mining frequent query patterns from xml queries. In *Proceedings of the 8th International Conference on Database Systems for Advanced Applications (DASFAA'03)*, 2003.

W. Yang, J. Dia, H. Cheng, and H. Lin. Mining social networks for targeted advertising. In *Proceedings of the 39th Annual Hawaii International Conference on System Science*, 2006.

Y. Yang and J.O. Pedersen. A comparative study on feature selection in text categorization. In *Proceedings of the 14th International Conference on Machine Learning*, pages 412–420, 1997.

S.H. Yook, H. Jeong, A.-L. Barabási, and Y. Tu. Weighted evolving networks. *Physical Review Letters*, 86:5835–5838, 2001.

U. Yun. Wis: Weighted interesting sequential pattern mining with a similar level of support and/or weight. *ETRI Journal*, 29(3):336–352, 2007.

U. Yun and J.J. Leggett. WFIM: Weighted frequent item-set mining with a weight range and a minimum weight. In *Proceedings of the 5th SIAM International Conference on Data Mining*, pages 636–640, 2005.

U. Yun and J.J. Leggett. WFIP: Mining weighted interesting patterns with a strong weight and/or support affinity. In *Proceedings of the 6th SIAM International Conference on Data Mining*, 2006.

M.J. Zaki. Efficiently mining frequent trees in a forest. In *Proceedings of the eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 71–80. ACM Press, New York, 2002.

M.J. Zaki. Efficiently mining frequent trees in a forest: algorithms and applications. *IEEE Transactions on Knowledge and Data Engineering*, 17(8):1021–1035, 2005a.

M.J. Zaki. Efficiently mining frequent embedded unordered trees. *Fundamenta Informaticae*, 66(1-2):33–52, 2005b.

M.J. Zaki and C.C. Aggarwal. Xrules: An effective structural classifier for xml data. In *Proceedings of the 2003 International Conference on Knowledge Discovery and Data Mining (SIGKDD'03)*, 2003.

M.J. Zaki and C.C. Aggarwal. Xrules: An effective structural classifier for xml data. *Machine Learning Journal*, 62(1-2):137–170, 2006.

M.J. Zaki and C.J. Hsiao. Charm: An efficient algorithm for closed itemset mining. In *Proceedings of the 2nd SIAM International Conference on Data Mining*, 2002.

Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Coherent closed quasi-clique discovery from large dense graph databases. In *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 797–802, 2006.

S. Zhang and J.T.L. Wang. Mining frequent agreement subtrees in phylogenetic databases. In *Proceedings of the 6th SIAM International Conference on Data Mining (SDM'06)*, pages 222–233, 2006.

S. Zhang and J. Yang. Ram: Randomized approximate graph mining. In *Proceedings of the 20th International Conference on Scientific and Statistical Database Management*, pages 187–203, 2008.

P. Zhao and J. Yu. Fast frequent free tree mining in graph databases. In *Proceedings of the 6th IEEE International Conference on Data Mining Workshop*, pages 315–319, 2006.

P. Zhao and J. Yu. Mining closed frequent free trees in graph databases. In *Proceedings of the 12th International Conference on Database Systems for Advanced Applications*, 2007.

F. Zhu, X. Yan, J. Han, and P. S. Yu. gPrune: A constraint pushing framework for graph pattern mining. In *Proceedings of 2007 Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD'07)*, pages 388–400, 2007.

Appendix A

Graph File Formats

For the work described in this thesis two graph file formats are employed: (i) a GraphML format, and (ii) a simple LineGraph format. The two representation formats are described in Sections A.1 and A.2 separately in the context of the example graph presented in Figure A.1.

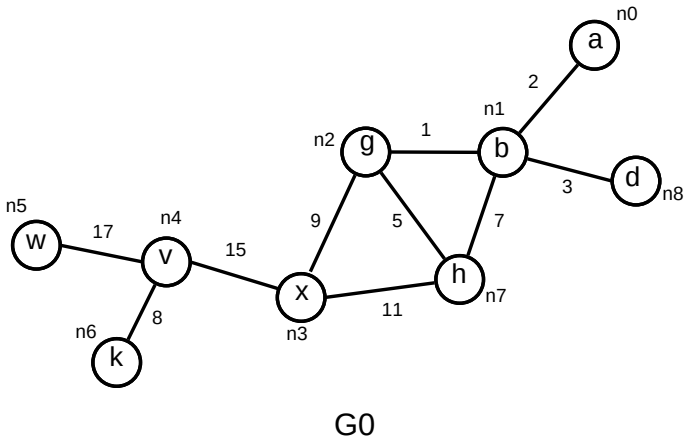


Figure A.1: A graph example

A.1 GraphML Format

GraphML is an XML based file format originally proposed by the graph community so as to provide a suitable file exchange format for graph data. The following presents an encoding of the graph $G0$ presented in Figure A.1 using GraphML.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <graphml xmlns="http://graphml.graphdrawing.org/xmlns"
3: xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4: xsi:schemaLocation="http://graphml.graphdrawing.org/xmlns
5:    http://graphml.graphdrawing.org/xmlns/1.0/graphml.xsd">
6:   <graph id="G0" edgedefault="undirected">
7:      <node id="n0"/><data>a</data></node>
```

```
8:      <node id="n1"/><data>b</data></node>
9:      <node id="n2"/><data>g</data></node>
10:     <node id="n3"/><data>x</data></node>
11:     <node id="n4"/><data>v</data></node>
12:     <node id="n5"/><data>w</data></node>
13:     <node id="n6"/><data>k</data></node>
14:     <node id="n7"/><data>h</data></node>
15:     <node id="n8"/><data>d</data></node>
16:     <edge id="e0" source="n0" target="n1"><data>2</data></edge>
17:     <edge id="e1" source="n1" target="n2"><data>1</data></edge>
18:     <edge id="e2" source="n1" target="n7"><data>7</data></edge>
19:     <edge id="e3" source="n1" target="n8"><data>3</data></edge>
20:     <edge id="e4" source="n2" target="n3"><data>9</data></edge>
21:     <edge id="e5" source="n2" target="n7"><data>5</data></edge>
22:     <edge id="e6" source="n3" target="n4"><data>15</data></edge>
23:     <edge id="e7" source="n3" target="n7"><data>11</data></edge>
24:     <edge id="e8" source="n4" target="n5"><data>17</data></edge>
25:     <edge id="e9" source="n4" target="n6"><data>8</data></edge>
26:  </graph>
27: </graphml>
```

Line 1 to 5 define a common header, line 7 to 15 define graph node information, and line 16 to 25 define graph edge information. In GraphML there is no required ordering for the expression of node and edge elements. More advanced features offered by the GraphML format can be found in the GraphML Primer[1].

A.2 Simple LineGraph Format

The simple LineGraph format is widely used by the researchers in the chemical informatics domain. Figure A.2 shows how to generate graphs using this format. In the figure, for each graph, the first line defines the graph id, this is followed by node definitions, and edge definitions. In the node definition (v1,v2,$\cdots$,vn) denote node ids, and (v1-lab,v2-lab,$\cdots$,vn-lab) denote node labels; in the edge definition, (s1,s2,$\cdots$,sn) and (t1,t2,$\cdots$,tn) denote source and target node ids connected by edges, and (e1-lab,e2-lab,$\cdots$,en-lab) denote edge labels.

A simple LineGraph representation for $G0$ graph given in Figure A.1 is as follows:

```
g # G0
v n0 a
v n1 b
v n2 g
v n3 x
v n4 v
v n5 w
v n6 k
v n7 h
```

[1] http://graphml.graphdrawing.org/primer/graphml-primer.html

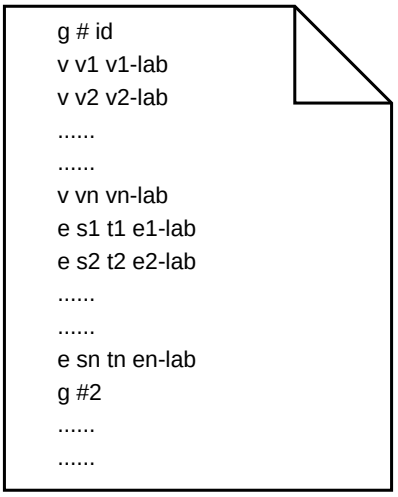


Figure A.2: A LineGraph format illustration

```
v n8 d
e n0 n1 2
e n1 n2 1
e n1 n7 7
e n1 n8 3
e n2 n3 9
e n2 n7 5
e n3 n4 15
e n3 n7 11
e n4 n5 17
e n4 n6 8
```

Appendix B

Additional Experimental Results

Some experimental results used for the evaluation of the subgraph weighting schemes, because of space restrictions, were omitted from the main body of this thesis. For completeness some of these additional results are presented here. These results are presented in Sections B.1, B.2, and B.3 respectively, according to the nature of the graph data: trees, undirected graphs, and graphs.

B.1 The Experimental Results for Trees

In this section, some results of testing five subgraph weighting schemes to the tree data are provided. The details with respect to each subgraph weighting scheme are delivered in the following sub-sections.

B.1.1 The application of the ATW scheme

B.1.1.1 The ST1, ST2, and RT1 data

The performance of the gSpan-ATW algorithm on the ST2 data is shown in Figure B.1. As can be seen in the figure, gSpan-ATW coupled with SW4 identifies more patterns than when coupled with SW1 or SW5.

B.1.1.2 The RT2 data

Figure B.2 illustrates the performance of gSpan-ATW with respect to the QT-D5 data set. From the figure it should be noted that gSpan, FFSM and GASTON fail to operate when using support thresholds of less than 30%, while gSpan-ATW, using either SW1 or SW4, operates with support thresholds down to 10%. The figure also indicates that gSpan-ATW, coupled with SW1 or SW4, requires less runtime and identifies far fewer patterns than gSpan, FFSM and GASTON. Further, the performance of gSpan-ATW using SW1 is very close to that when using SW4.

The performance of gSpan-ATW when applied to the QT-D6 data set is shown in Figure B.3. It should be noted that in this case the gSpan, FFSM and GASTON

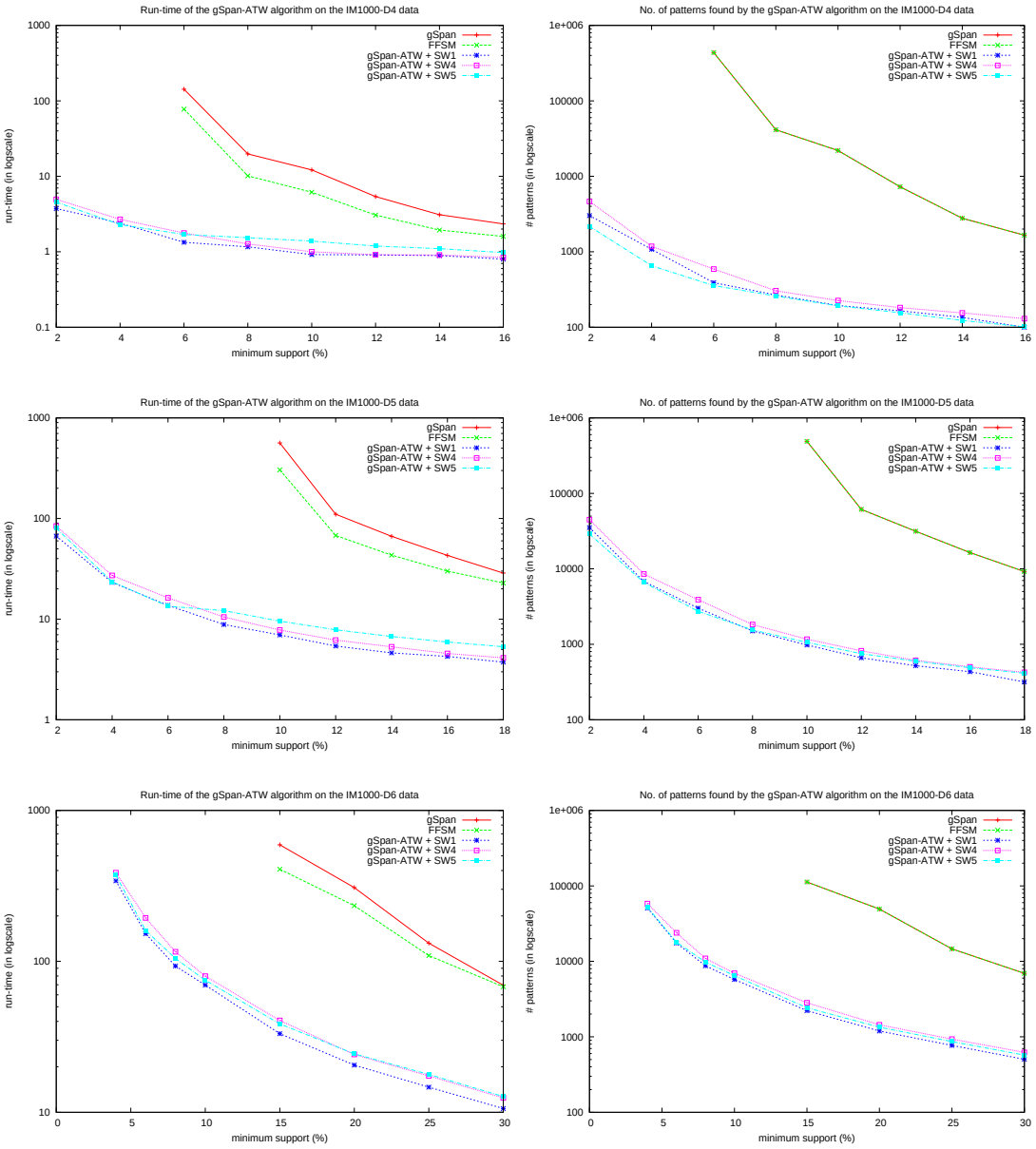


Figure B.1: The performance of gSpan-ATW on the ST2 data

algorithms fail to operate with support thresholds of less than 35% because of the out-of-memory error. The gSpan-ATW algorithm, however, can operate using support thresholds of less than 15%. As can be seen from the figure, gSpan-ATW outperforms gSpan, FFSM and GASTON, with respect to runtime and the number of patterns discovered. Additionally, gSpan-ATW, when coupled with SW1, runs slightly faster than when using SW4; both algorithms discover very similar numbers of patterns.

The performance of gSpan-ATW on the QT-D7 data is shown in Figure B.4. In the figure, the curves representing the FFSM and gSpan algorithms stop at a support threshold of 45% (GASTON fails to operate due to the out-of-memory error), while

Table B.1: The performance of gSpan-ATW using SW4 and SW5 on the ST1, ST2, and RT1 data

Dataset	τ(%)	gSpan-ATW + SW4		v.	gSpan-ATW + SW5	
		runtime (in seconds)	# patterns		runtime (in seconds)	# patterns
ST1:D10	0.05	4.167	134		5.194	346
	0.1	3.903	95		4.992	239
	0.5	3.680	42		4.165	67
	1	3.631	32		3.946	59
	2	2.870	22		3.884	31
ST1:T1M	0.05	114.005	118		230.203	414
	0.1	106.209	81		211.472	225
	0.5	99.854	41		159.839	66
	1	92.626	32		157.080	59
	2	94.357	21		150.138	30
ST2:IM1000-D4	6	1.595	590		1.695	357
	8	1.269	304		1.535	259
	10	1.000	226		1.384	192
	12	0.914	182		1.191	156
	14	0.906	155		1.101	123
ST2:IM1000-D5	10	7.795	1169		9.565	1059
	12	6.191	820		7.829	751
	14	5.310	613		6.726	597
	16	4.557	502		5.921	489
	18	4.127	426		5.315	414
ST2:IM1000-D6	15	38.379	2822		38.536	2426
	20	24.168	1444		24.390	1341
	25	17.426	932		17.823	864
	30	12.462	622		12.728	572
RT1:CSLOGS-ALL	0.3	3.366	617		4.612	616
	0.4	3.176	425		4.382	424
	0.5	3.040	304		4.179	303
	0.6	2.963	244		3.970	243
	0.8	2.882	169		3.904	169
RT1:CSLOGS-1	0.2	0.733	631		0.779	621
	0.4	0.642	286		0.775	287
	0.6	0.616	176		0.741	176
	0.8	0.608	135		0.695	135
	1	0.597	106		0.664	105
RT1:CSLOGS-2	0.2	0.717	674		0.774	662
	0.4	0.679	293		0.711	292
	0.6	0.633	180		0.681	180
	0.8	0.598	135		0.673	135
	1	0.556	103		0.641	102

the two curves representing the gSpan-ATW algorithms (coupled with SW1 and SW4 respectively) continue to operate down to a support threshold of 25%. Clearly gSpan-ATW runs faster and identifies fewer patterns than both gSpan and FFSM. In addition, gSpan-ATW coupled with SW1 outperforms gSpan-ATW couple with SW4, in terms of the runtime cost and the number of patterns discovered.

B.1.2 The application of the AW scheme

B.1.2.1 The ST1, ST2, and RT1 data

The AW scheme is only applicable to the ST1, ST2, RT1:CSLOGS-1, and RT1:CSLOGS-2 data sets. Thus, the experimental results on these data sets are presented in this

Table B.2: The accuracy of the classifiers using patterns discovered by gSpan-ATW with SW4 and SW5 on the ST2 and RT1:CSLOGS-1(2) data

Dataset	gSpan-ATW + SW4					gSpan-ATW + SW5				
	τ(%)	#F	NBC	SVM	C4.5	τ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	2	4646	92.9	95.4	94.9	2	2173	92.8	95.6	94.9
	4	1182	93.0	95.6	94.3	4	653	93.0	95.6	94.3
	6	590	93.3	95.4	93.7	6	357	93.6	95.7	93.8
ST2:IM1000-D5	6	3897	86.2	91.4	91.3	6	2740	86.2	91.3	91.4
	8	1829	86.2	91.4	91.6	8	1550	86.2	91.1	91.4
	10	1169	85.8	91.4	89.7	10	1059	85.2	88.7	91.4
ST2:IM1000-D6	12	4918	81.2	75.1	86.5	12	4193	81.2	75.1	86.5
	14	3475	81.2	75.1	86.5	14	2852	81.2	75.1	86.5
	16	2377	81.2	75.1	86.5	16	2126	81.2	75.1	86.5
RT1:CSLOGS-1	0.2	631	79.8	81.1	80.9	0.2	621	79.8	81.1	80.9
	0.3	394	79.8	81.1	80.9	0.3	391	79.8	81.1	80.9
	0.4	286	79.8	80.9	80.9	0.4	287	79.8	80.9	80.9
RT1:CSLOGS-2	0.2	674	80.4	81.0	81.5	0.2	662	80.4	81.0	81.5
	0.3	384	80.4	81.7	81.5	0.3	383	80.4	81.7	81.5
	0.4	293	80.4	81.6	81.5	0.4	292	80.4	81.6	81.5

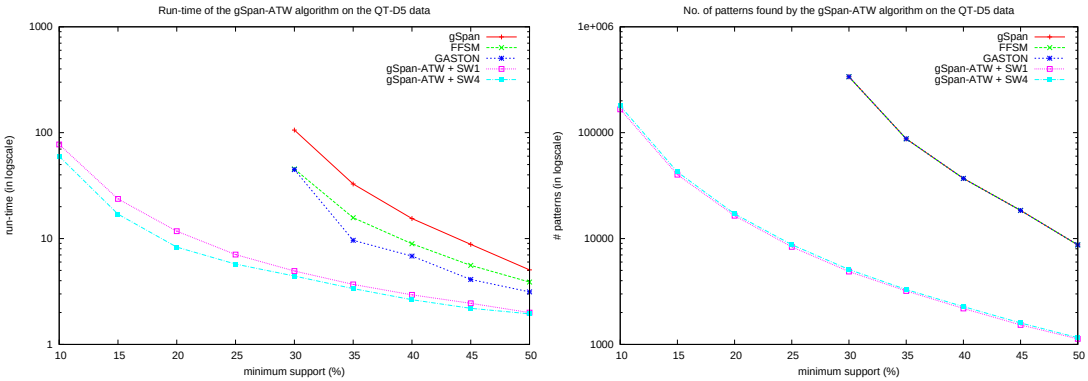


Figure B.2: The performance of gSpan-ATW on the QT-D5 data

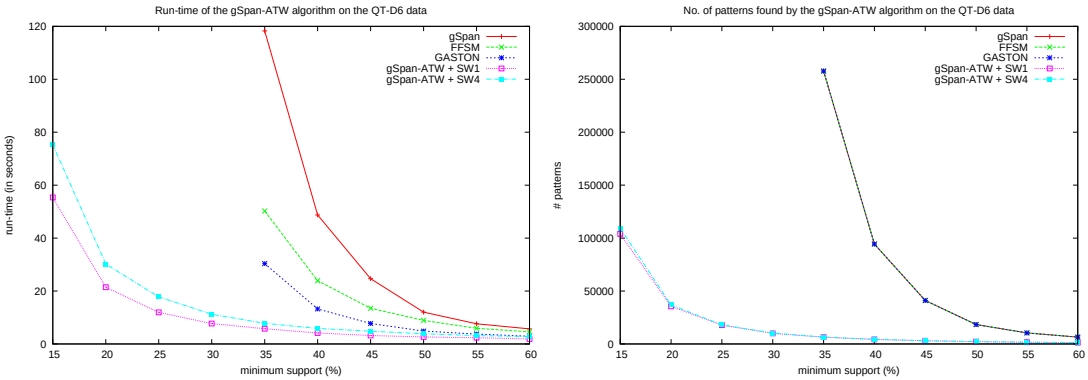


Figure B.3: The performance of gSpan-ATW on the QT-D6 data

sub-section.

For the IM1000-D4 data set, Figure B.5 shows that gSpan-AW coupled with either

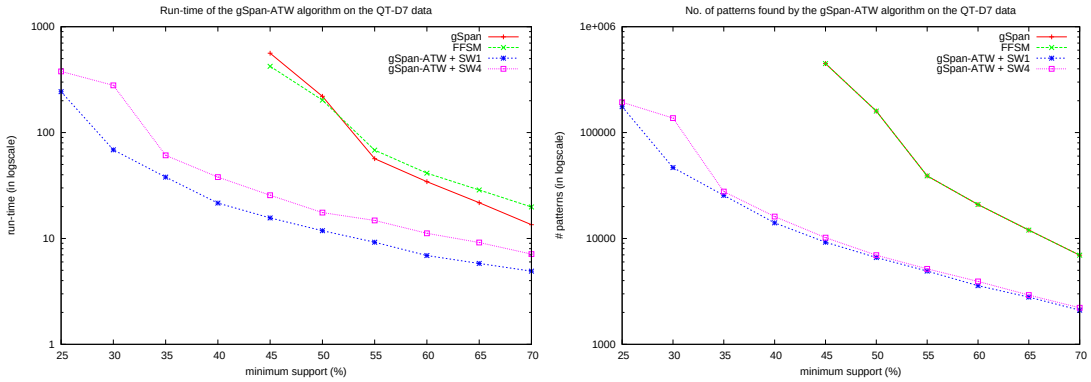


Figure B.4: The performance of gSpan-ATW on the QT-D7 data

Table B.3: The performance of gSpan-AW using SW4 on the ST1, ST2, and RT1:CSLOGS-1(2) data

Dataset	λ	τ(%)	gSpan-AW + SW4	
			runtime (in seconds)	# patterns
ST1:D10	0.01%	0.05	4.183	121
		0.1	3.849	93
		0.5	3.592	35
		1	3.517	25
		2	3.122	19
ST1:T1M	0.01%	0.05	102.869	88
		0.1	98.099	69
		0.5	95.496	30
		1	93.139	22
		2	89.644	18
ST2:IM1000-D4	0.02	6	1.370	189
		8	1.247	134
		10	1.091	110
		12	1.053	98
		14	1.002	84
ST2:IM1000-D5	0.06	10	4.889	519
		12	3.888	350
		14	3.454	279
		16	3.001	239
		18	2.880	202
ST2:IM1000-D6	0.06	15	8.068	396
		20	6.300	255
		25	5.283	187
		30	4.449	139
RT1:CSLOGS-1	0.6	0.2	0.692	718
		0.4	0.650	300
		0.6	0.617	177
		0.8	0.604	134
		1	0.593	104
RT1:CSLOGS-2	0.6	0.2	0.750	819
		0.4	0.604	311
		0.6	0.568	185
		0.8	0.565	135
		1	0.540	103

SW1 or SW4 performs better than both gSpan and FFSM, in terms of the runtime cost and the number of patterns discovered. Moreover, gSpan-AW coupled with SW4

identifies fewer patterns with slightly higher runtime costs than when using SW1.

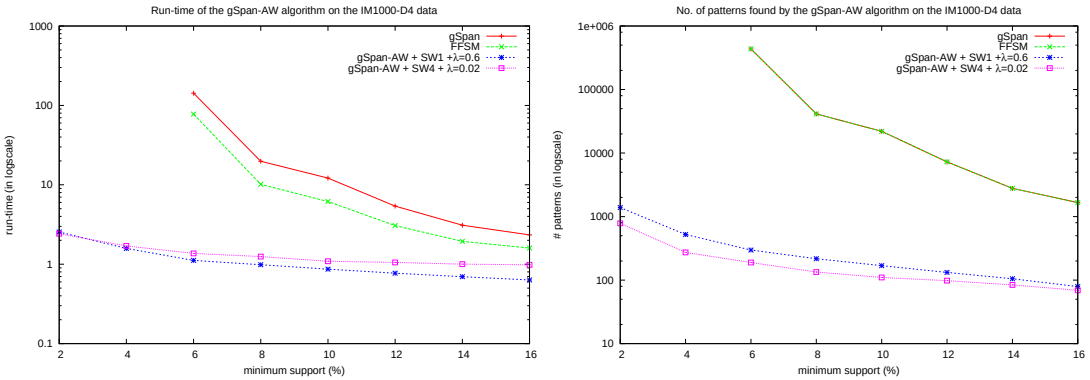


Figure B.5: The performance of gSpan-AW on the IM1000-D4 data

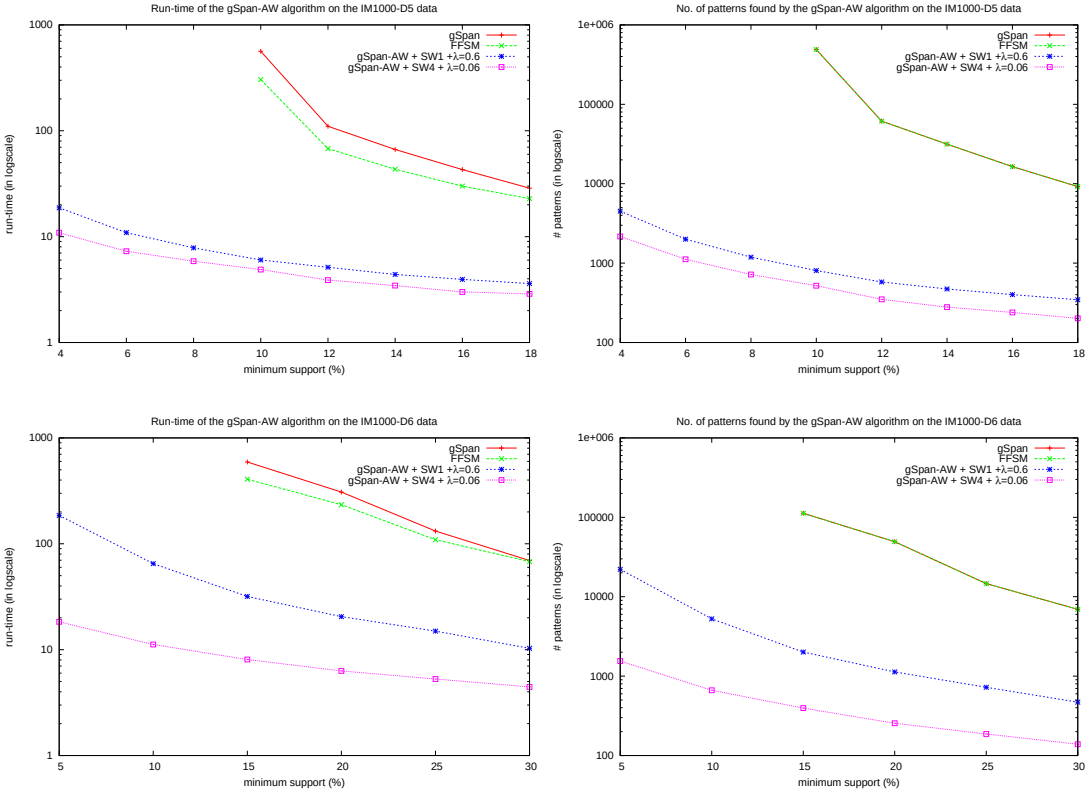
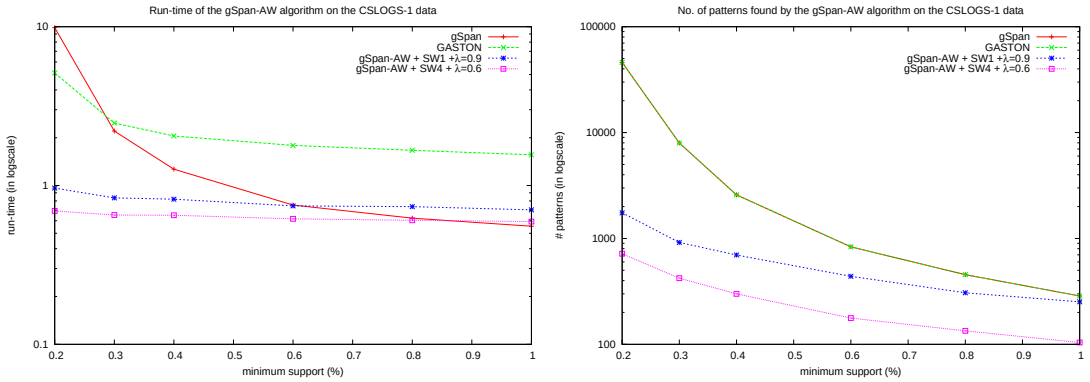


Figure B.6: The performance of gSpan-AW on the IM1000-D5 and IM1000-D6 data

For data sets IM1000-D5 and IM1000-D6, the performance of gSpan-AW is presented in Figure B.6. In the figure, the performance of gSpan-AW on IM1000-D5 is similar to that of gSpan-AW on IM1000-D6. All the curves in the figure indicate that the performance of gSpan-AW coupled with either SW1 or SW4 identifies fewer patterns with lower runtime costs than gSpan and FFSM. Further, for each of these two data sets, gSpan-AW coupled with SW4 performed better than when using SW1.

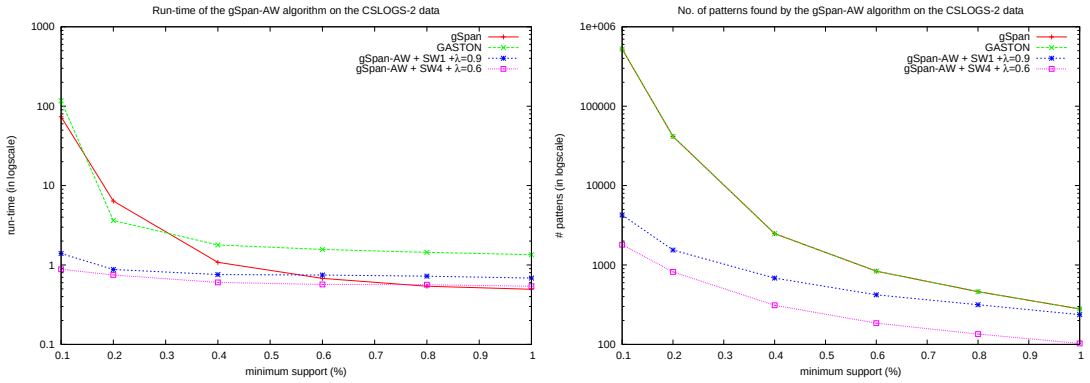


(a) (b)

Figure B.7: The performance of gSpan-AW on the CSLOGS-1 data

The performance of gSpan-AW on the CSLOGS-1 data set is shown in Figure B.7. Figure B.7 (a) indicates that the GASTON algorithm is the slowest when using support thresholds of between 0.3% and 1%; gSpan-AW, coupled with SW1, runs slower than gSpan when the support threshold is over 0.6% and the former runs faster than the latter when the support threshold is below 0.6%. Figure B.7 (b) also shows that gSpan-AW identifies fewer patterns than the standard FSM algorithms. In addition, gSpan-AW, when coupled with SW4, appears to run faster and identifies fewer patterns than when using SW1.

The performance of gSpan-AW on the CSLOGS-2, which is shown in Figure B.8, exhibits similar behaviour to that shown in Figure B.8.



(a) (b)

Figure B.8: The performance of gSpan-AW on the CSLOGS-2 data

B.1.2.2 The RT2 data

The performance of the gSpan-AW algorithm on the RT2 data is presented in Figure B.9. It can be seen from the figure that gSpan-AW discovers substantially fewer patterns

Table B.4: The accuracy of the classifiers using patterns discovered by gSpan-AW with SW4 on the ST2 and RT1:CSLOGS-1(2) data

Dataset	gSpan-AW+ SW4					
	λ	τ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	2%	2	784	89.2	95.2	95.1
		4	273	89.6	95.2	95.1
		6	189	86.4	95.8	94.9
ST2:IM1000-D5	6%	4	2167	82.7	87.4	86.4
		6	1118	82.7	87.4	86.4
		8	719	82.7	87.6	86.1
ST2:IM1000-D6	6%	10	663	80.2	76.7	85.0
		15	396	81.1	77.1	82.9
		20	255	80.9	75.7	83.8
RT1:CSLOGS-1	0.6	0.2	718	80.1	81.9	82.3
		0.4	300	80.1	81.7	82.2
		0.6	177	79.9	81.3	81.8
RT1:CSLOGS-2	0.6	0.1	1801	80.5	79.5	82.2
		0.2	819	80.5	81.1	82.2
		0.4	311	80.5	81.5	82.0

using significantly less runtime than gSpan, FFSM and GASTON.

However, compared with the performance of the classifiers built using patterns discovered by gSpan as shown in Table B.5, Table B.6 suggests that the patterns discovered by gSpan-AW are not as effective as those discovered by gSpan, in terms of the accuracy of the classifiers.

Table B.5: The accuracy of the classifiers using patterns discovered by the standard FSM algorithms on the RT2 data

Dataset	standard FSM algorithms				
	σ(%)	#F	NBC	SVM	C4.5
RT2:QT-D5	35	87540	76.4	73.6	66.0
	40	36955	79.2	76.4	68.9
	45	18448	73.6	69.8	56.6
	50	8715	75.5	66.0	55.7
RT2:QT-D6	45	41096	85.8	84.9	73.6
	50	18353	82.1	85.8	78.3
	55	10423	80.2	79.2	74.5
	60	6438	81.1	80.2	73.6
RT2:QT-D7	55	39008	82.1	77.4	62.3
	60	20905	80.2	78.3	63.2
	65	11998	80.2	77.4	69.8
	70	6959	81.1	77.4	70.8

B.1.3 The application of the CMW scheme

According to whether or not the tree data features class labels, the experimental results of applying the CMW scheme to the tree data are presented in the following two subsections.

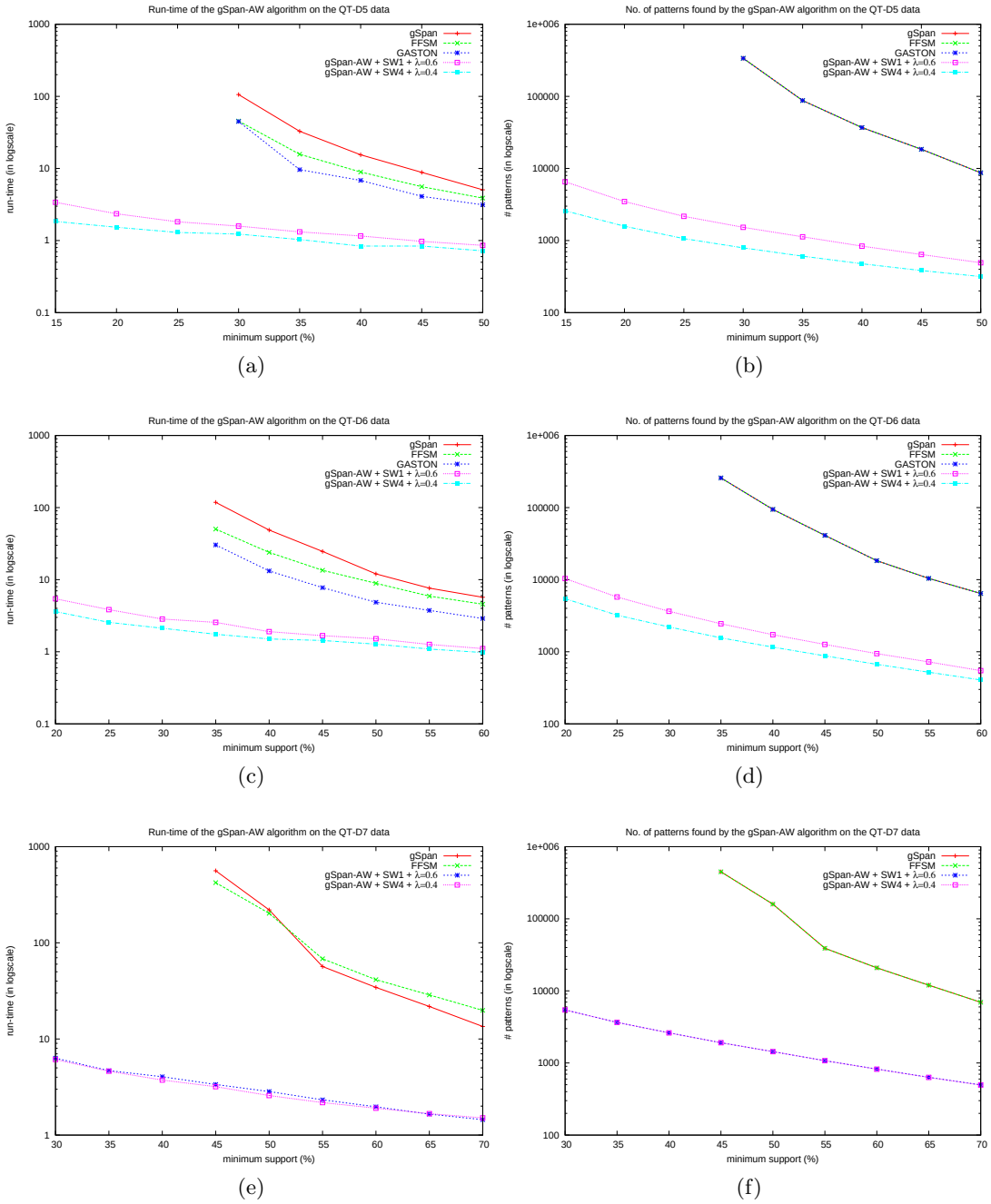


Figure B.9: The performance of gSpan-AW on the RT2 data

B.1.3.1 The ST1 and RT1:CSLOGS-ALL data

Because the trees in ST1 and RT1:CSLOGS-ALL do not have class labels, the gSpan-CMW algorithm was coupled with the SW2 or SW3 function. The performance of gSpan-CMW with SW3 on the ST1 and RT1:CSLOGS-ALL data is displayed in Table B.7. In comparison with the performance of gSpan-CMW with SW2 on the same groups of data, as shown in Table 6.18, Table B.7 suggests that gSpan-CMW coupled

Table B.6: The accuracy of the classifiers using patterns discovered by gSpan-AW with SW1 and SW4 on the RT2 data

Dataset	gSpan-AW + SW1						gSpan-AW + SW4					
	λ	τ(%)	#F	NBC	SVM	C4.5	λ	τ(%)	#F	NBC	SVM	C4.5
RT2:QT-D5	0.6	15	6537	70.8	55.7	57.5	0.4	15	2587	57.5	53.8	53.8
		20	3478	68.9	56.6	58.5		20	1582	54.7	52.8	56.6
		25	2169	60.4	53.8	59.4		25	1065	55.7	52.8	59.4
RT2:QT-D6	0.6	25	5753	66.0	59.4	67.0	0.4	15	9794	79.2	68.9	64.2
		30	3638	66.0	57.5	67.0		20	5444	69.8	71.7	61.3
		35	2448	68.9	64.2	67.0		25	3219	69.8	63.2	57.5
RT2:QT-D7	0.6	20	14564	70.8	69.8	72.6	0.4	25	8663	72.6	75.5	66.0
		25	8665	72.6	75.5	66.0		30	5437	69.8	72.6	64.2
		30	5439	67.0	71.7	66.0		35	3659	63.2	72.6	73.6

with SW3 is more efficient than that coupled with SW2, in terms of the runtime cost and the number of patterns discovered.

Table B.7: The performance of gSpan-CMW using SW3 on the ST1 and RT1:CSLOGS-ALL data

Dataset	θ	σ(%)	gSpan-CMW + SW3	
			runtime (in seconds)	# patterns
ST1:D10	0.8	0.05	3.863	96
		0.1	3.254	76
		0.5	2.437	31
		1	2.203	23
		2	2.139	22
ST1:T1M	0.6	0.05	63.735	84
		0.1	58.326	66
		0.5	46.735	31
		1	43.869	23
		2	43.750	21
RT1:CSLOGS-ALL	0.6	0.3	2.711	650
		0.4	2.282	457
		0.5	2.115	328
		0.6	1.967	265
		0.8	1.713	181

Table B.8: The performance of gSpan-CMW using CW3 on the ST2 data

Dataset	θ	σ(%)	gSpan-CMW + CW3	
			runtime (in seconds)	# patterns
ST2:IM1000-D4	0.4%	6	0.765	175
		8	0.730	140
		10	0.650	108
		12	0.573	82
		14	0.563	72
ST2:IM1000-D5	0.4%	10	1.459	280
		12	1.313	212
		14	1.177	167
		16	1.152	136
		18	1.070	116
ST2:IM1000-D6	0.4%	15	2.734	295
		20	2.307	229
		25	1.964	140
		30	1.757	81

B.1.3.2 The ST2 and RT1:CSLOGS-1(2) data

Since the trees in ST2 and RT1:CSLOGS-1(2) have class labels, the gSpan-CMW algorithm was coupled with the CW2 and CW3 functions. The performance of gSpan-CMW with CW3 on the ST2 and RT1:CSLOGS-1(2) data is shown in Tables B.8 and B.9 respectively.

Table B.9: The performance of gSpan-CMW using CW3 on the RT1:CSLOGS-1(2) data

Dataset	θ	σ(%)	gSpan-CMW + CW3	
			runtime (in seconds)	# patterns
RT1:CSLOGS-1	1%	0.2	1.716	629
		0.4	1.046	293
		0.6	0.827	183
		0.8	0.702	138
		1	0.655	107
RT1:CSLOGS-2	1%	0.2	1.507	678
		0.4	0.962	303
		0.6	0.749	188
		0.8	0.640	138
		1	0.612	105

Table B.10: The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW3 on the ST2 and RT1:CSLOGS-1(2) data

Dataset	gSpan-CMW+ CW3					
	θ	σ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	0.4%	2	666	89.6	95.0	95.0
		4	334	89.6	94.9	95.2
		6	175	89.6	94.9	95.2
ST2:IM1000-D5	0.4%	2	1514	86.9	88.0	91.3
		4	834	86.1	88.6	91.0
		6	559	85.9	88.4	90.7
ST2:IM1000-D6	0.4%	2	752	82.8	75.3	86.1
		5	490	90.9	74.6	85.5
		10	295	82.6	74.8	84.4
RT1:CSLOGS-1	1%	0.1	1284	80.7	81.1	82.1
		0.2	629	80.7	81.8	82.1
		0.3	398	80.7	81.7	82.1
RT1:CSLOGS-2	1%	0.1	1342	80.4	81.1	82.6
		0.2	678	80.4	81.5	82.6
		0.3	394	80.4	81.4	82.5

B.1.4 The application of the JSW scheme

The experimental results of applying the JSW scheme to the tree data are presented in the following sub-sections for ST1, ST2, RT1, and RT2.

B.1.4.1 The ST1 data

Table B.11: The performance of gSpan-JSW with different γ values on the ST1 data

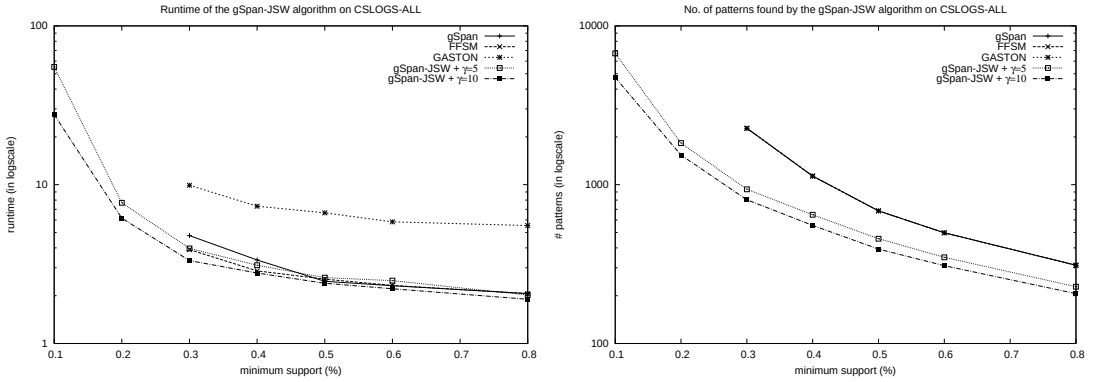
Dataset	γ	σ(%)	gSpan-JSW	
			runtime (in seconds)	# patterns
ST1:D10	5	0.05	14.539	201
		0.1	8.860	127
		0.5	5.039	42
		1	3.244	28
		2	3.166	26
ST1:T1M	8	0.05	193.961	145
		0.1	136.942	101
		0.5	66.010	32
		1	61.548	23
		2	60.095	20

B.1.4.2 The ST2 data

The performance of gSpan-JSW with two different γ values on the ST2:IM1000-D4, ST2:IM1000-D5, and ST2:IM1000-D6 data sets, in comparison with gSpan and FFSM, is depicted in Figure B.10. As can be seen in the figure, gSpan-JSW, with a larger γ value, runs much faster and discovers far fewer patterns than when a smaller γ value is used.

B.1.4.3 The RT1 data

Figure B.11 shows the performance of gSpan-JSW using different γ values on the CSLOGS-ALL data. From the figure it can be seen that the three standard FSM algorithms can only work successfully at support thresholds of over 0.3% while gSpan-JSW operates down to a support threshold of 0.1%. Apparently, Figure B.11 indicates that gSpan-JSW with a larger γ value is more efficient than that with a smaller γ value, in terms of the runtime cost and the number of patterns discovered.



(a) (b)

Figure B.11: The performance of gSpan-JSW on the CSLOGS-ALL data

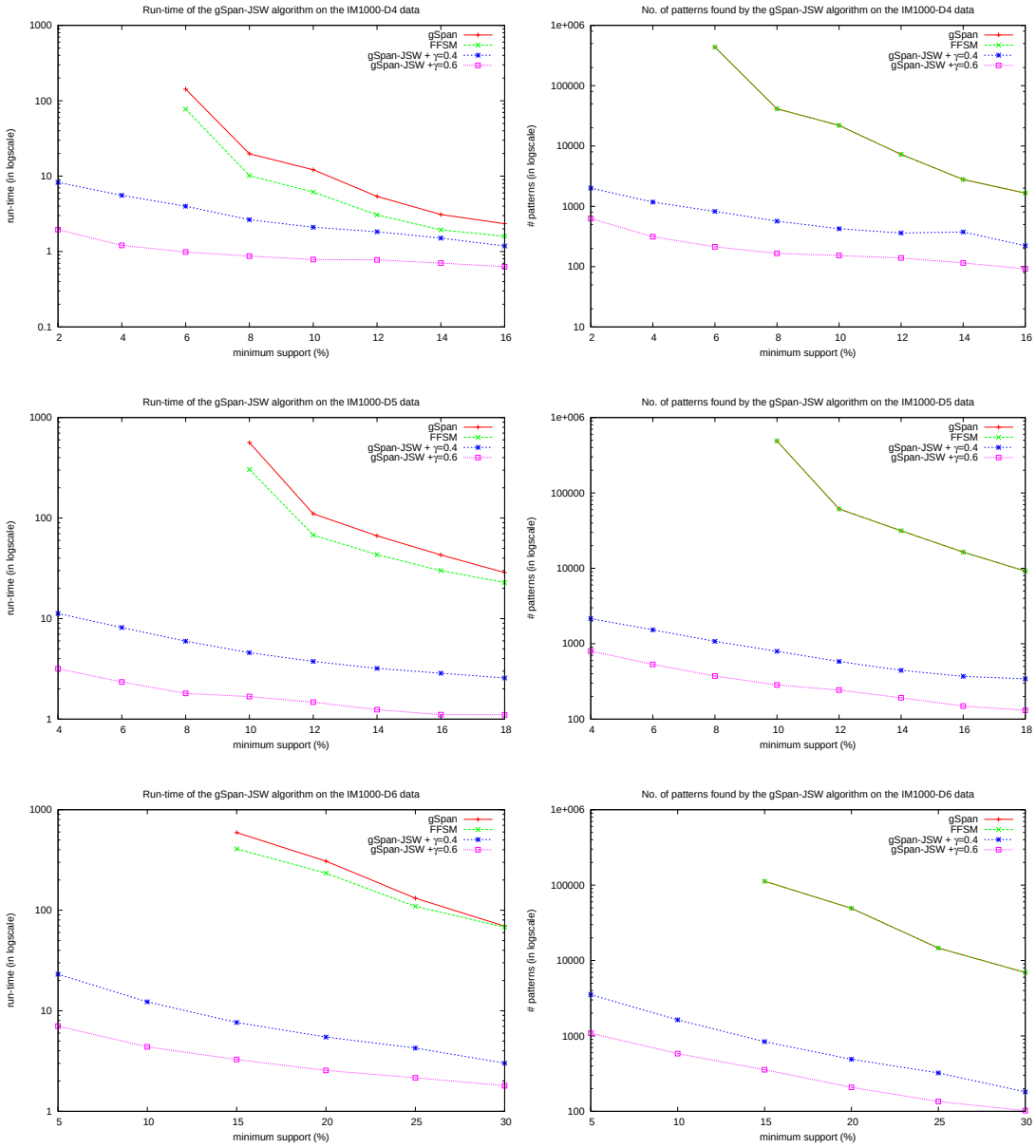


Figure B.10: The performance of gSpan-JSW on the ST2 data

When gSpan-JSW was applied to the RT1:CSLOGS-1 and RT1:CSLOGS-2 data sets, the performance of gSpan-JSW with $\gamma = 5$ is presented in Table B.12.

B.1.4.4 The RT2 data

The performance of gSpan-JSW using $\gamma = 0.25$ on the RT2 data is shown in Table B.13. In comparison with the performance of gSpan-JSW using a smaller γ value, as shown in Table 7.4, Table B.13 suggests that gSpan-JSW with a larger γ value runs faster and discovers fewer patterns than that with a smaller γ value.

Table B.12: The performance of gSpan-JSW with $\gamma = 5$ on the CSLOGS-1(2) data

Dataset	γ	σ(%)	gSpan-JSW	
			runtime (in seconds)	# patterns
RT1:CSLOGS-1	5	0.2	1.610	999
		0.4	0.908	417
		0.6	0.731	242
		0.8	0.612	177
		1	0.559	137
RT1:CSLOGS-2	5	0.2	1.581	1042
		0.4	0.893	429
		0.6	0.665	247
		0.8	0.596	186
		1	0.560	140

Table B.13: The performance of gSpan-JSW with $\gamma = 0.25$ on the RT2 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-JSW			
		gSpan	FFSM	GASTON		γ	σ(%)	runtime	# patterns
RT2:QT-D5	30	105.793	45.426	44.838	337242	0.25	30	3.396	2797
	35	32.754	15.728	9.620	87540		35	2.940	2406
	40	15.450	8.923	6.834	36955		40	2.626	2085
	45	8.818	5.583	4.106	18448		45	2.391	1796
	50	5.061	3.881	3.136	8715		50	2.169	1556
RT2:QT-D6	35	118.280	50.243	30.360	257811	0.25	35	4.294	2811
	40	48.741	23.893	13.234	94333		40	3.810	2531
	45	24.653	13.499	7.751	41096		45	3.498	2270
	50	12.025	8.896	4.868	18353		50	3.172	2002
	55	7.630	5.913	3.745	10423		55	2.800	1729
RT2:QT-D7	45	561.675	422.965	n/a	448683	0.25	45	7.091	2927
	50	219.924	202.759	n/a	159507		50	6.468	2671
	55	56.686	68.235	n/a	39008		55	5.856	2399
	60	34.394	41.410	n/a	20905		60	5.581	2126

B.1.5 The application of the UBW scheme

The performance of gSpan-UBW on the CSLOGS-ALL data set is presented in Figure B.12. In the figure, both gSpan and FFSM stop when a support threshold of 0.3% is reached, while gSpan-UBW continues down to a support threshold of 0.1%. As can be seen from the figure, both gSpan and FFSM run slightly faster than gSpan-UBW when the support threshold is between 0.3% and 1%, but the number of patterns discovered by the former is considerably more than that discovered by the latter. Moreover, gSpan-UBW using the SW1 function runs moderately faster than that using SW4.

Figure B.13 illustrates the performance of gSpan-UBW on the CSLOGS-1 data set. In the figure, when the support threshold is between 0.3% and 1%, both gSpan and GASTON run faster than gSpan-UBW. When the support threshold falls to below 0.2% both gSpan and FFSM can not proceed with the mining due to the out-of-memory errors, while gSpan-UBW continues down to a support threshold of 0.1%. Because the runtime for all the algorithms is within 10 seconds, it is not very useful to compare the runtime. However, it is of interest to see in Figure B.13 (b) that the number of patterns identified by gSpan-UBW is considerably less than that discovered by the

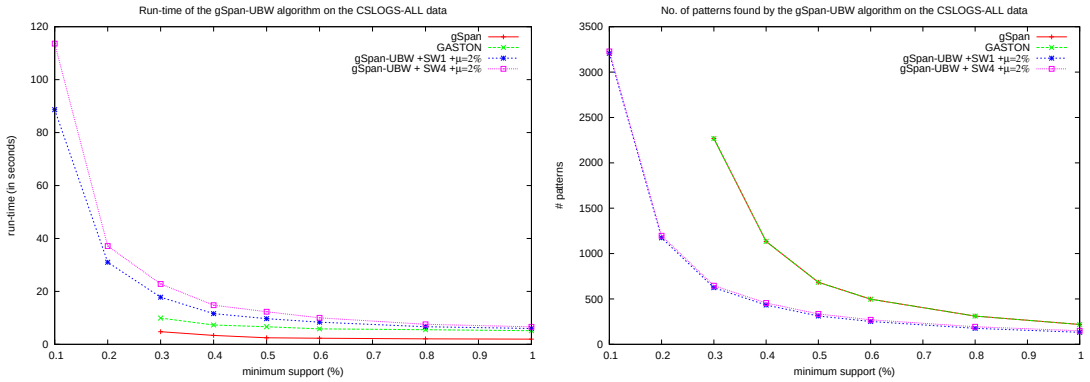
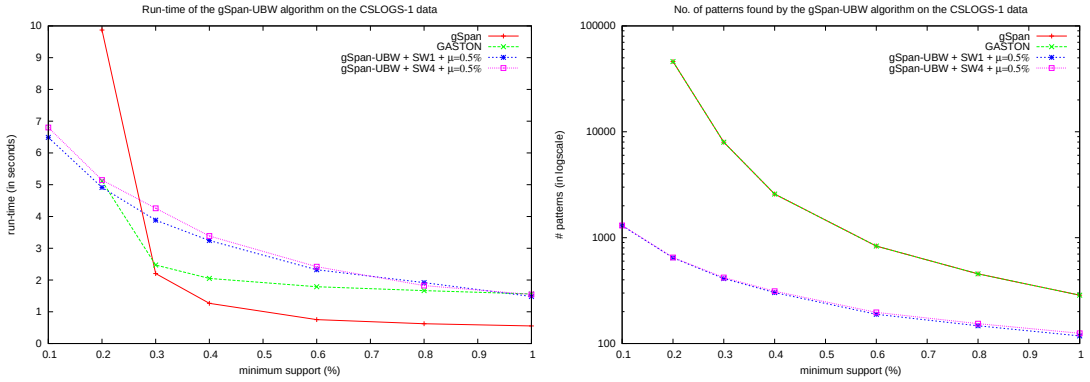


Figure B.12: The performance of gSpan-UBW on the CSLOGS-ALL data

standard FSM algorithms.



(a) (b)

Figure B.13: The performance of gSpan-UBW on the CSLOGS-1 data

The performance of gSpan-UBW on the CSLOGS-2 data set is shown in Figure B.14. In the figure, both gSpan and GASTON mostly run faster than gSpan-UBW, but the former run slower than the latter when the support threshold is below 0.2%. Again, the big gap (in Figure B.14 (b)) between the number of patterns identified by the standard FSM algorithms and that identified by gSpan-UBW indicates that gSpan-UBW cuts down significantly on the number of patterns discovered by gSpan.

The performance of gSpan-UBW on the QT-D5 data set is presented in Figure B.15. In the figure, gSpan, FFSM and GASTON stop when the support threshold reaches 30% while gSpan-UBW continues until the support threshold reaches 10%. It can also be seen from the figure that gSpan-UBW runs faster and discovers far fewer patterns than gSpan, FFSM and GASTON.

Figure B.16 illustrates the performance of the gSpan-UBW algorithm on the QT-D6 data set. In the figure, gSpan, FFSM and GASTON stop when the support threshold reaches 35%, while gSpan-UBW continues until a support threshold of 10% is reached.

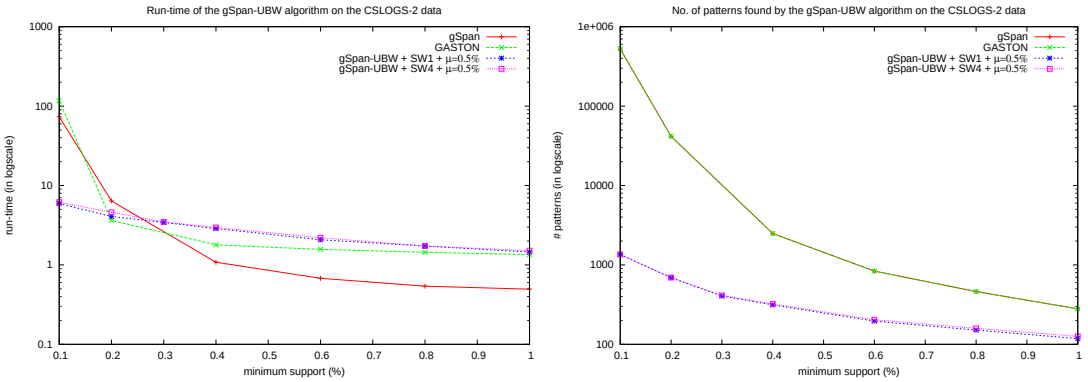


Figure B.14: The performance of gSpan-UBW on the CSLOGS-2 data

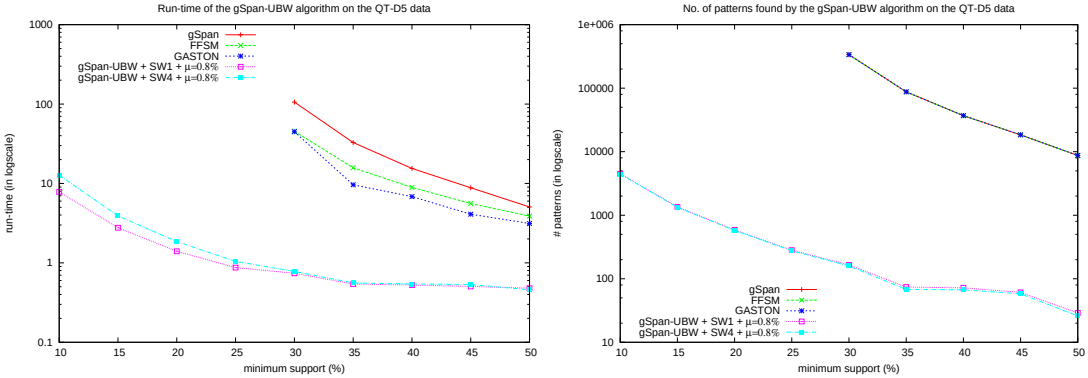


Figure B.15: The performance of gSpan-UBW on the QT-D5 data

From the figure it can be seen that gSpan-UBW clearly outperforms both gSpan, FFSM and GASTON, in terms of runtime and number of patterns discovered.

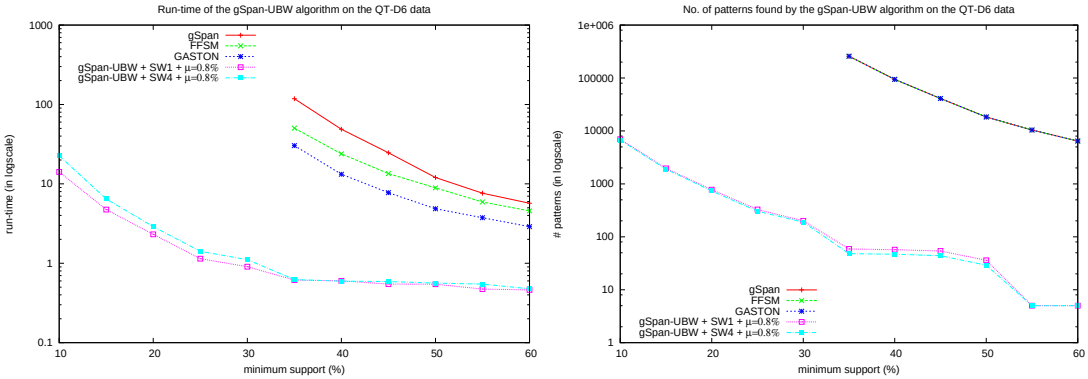


Figure B.16: The performance of gSpan-UBW on the QT-D6 data

The performance of gSpan-UBW on the QT-D7 data set is shown in Figure B.17. In the figure, both gSpan and FFSM stop when a support threshold of 45% is reached,

while gSpan-UBW continues until a support threshold of 15% is reached. As can be seen from the figure, gSpan-UBW identifies substantially fewer patterns and requires significantly less runtime than both gSpan and FFSM.

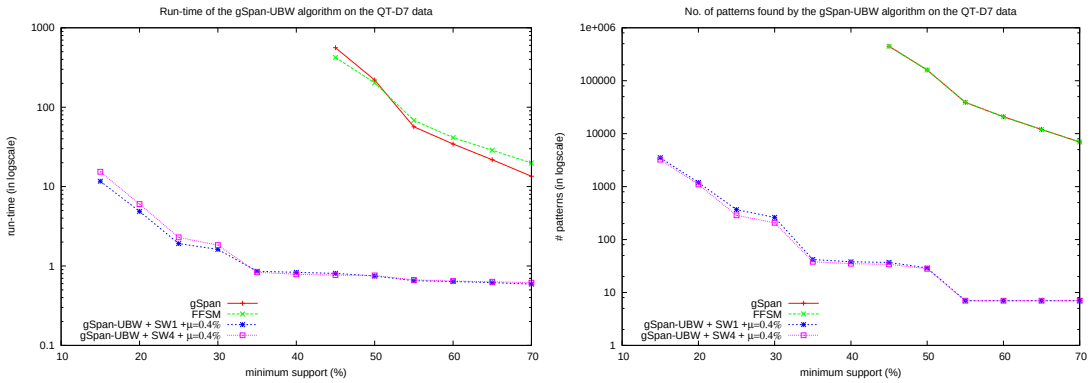


Figure B.17: The performance of gSpan-UBW on the QT-D7 data

Table B.14: The accuracy of the classifiers using patterns discovered by gSpan-UBW with SW4 on the tree data

Dataset	gSpan-UBW + SW4					
	μ	τ(%)	#F	NBC	SVM	C4.5
ST2:IM1000-D4	0.2%	6	421	93.3	95.4	93.8
		8	270	93.5	95.3	93.9
		10	179	92.3	95.7	94.4
ST2:IM1000-D5	0.2%	6	1567	90.3	92.9	92.3
		8	787	86.1	92.0	91.9
		10	469	84.3	91.9	90.0
ST2:IM1000-D6	0.2%	8	1964	85.8	79.6	89.8
		10	1106	85.2	79.3	89.8
		12	645	83.6	76.2	88.8
RT1:CSLOGS-1	0.5%	0.1	1304	80.6	81.2	82.1
		0.2	650	80.4	81.8	82.1
		0.4	312	80.6	81.7	82.1
RT1:CSLOGS-2	0.5%	0.1	1355	80.6	80.7	82.4
		0.2	694	80.6	81.3	82.2
		0.4	322	80.6	82.4	82.2
RT2:QT-D5	0.8%	8	8691	81.1	68.9	73.6
		10	4450	75.5	69.8	73.6
		12	2561	73.6	67.9	80.2
RT2:QT-D6	0.8%	10	6713	84.9	79.2	82.1
		12	3775	80.2	76.4	71.7
		14	2286	80.2	72.6	77.4
RT2:QT-D7	0.4%	12	7420	74.5	75.5	76.4
		14	4175	72.6	78.3	75.5
		16	2517	67.0	70.8	68.9
RT3	0.2%	15	1539	95.0	73.0	90.5
		20	812	94.5	63.0	92.5
		25	515	92.0	69.0	90.5

B.2 The Experimental Results for Undirected Graphs

Three groups of undirected graph data: RG1, RG2, and RG3 were used for the experiments. Some of the results of using different subgraph weighting schemes on these data are presented in the following sub-sections.

B.2.1 The application of the ATW scheme

B.2.1.1 The RG1 data

Table B.15: The performance of gSpan-ATW using SW4 and SW5 on the RG1 data

Dataset	τ(%)	gSpan-ATW + SW4		vs.	gSpan-ATW + SW5	
		runtime (in seconds)	# patterns		runtime (in seconds)	# patterns
RG1:CH1	10	270.430	797		219.518	586
	12	229.750	639		198.907	483
	14	199.295	529		170.474	395
	16	177.647	450		153.259	339
RG1:CH2	4	89.019	402		78.988	322
	6	69.278	287		65.318	229
	8	60.510	218		51.668	160
	10	51.164	163		43.218	126

The performance of gSpan-ATW on the CH1 data is presented in Figure B.18. It can be seen from the figure that the three curves representing the operation of gSpan-ATW coupled with SW1, SW4 and SW5 fall below the curve representing the operation of gSpan, and gSpan-ATW continues to operate down to a support threshold of 1% and beyond.

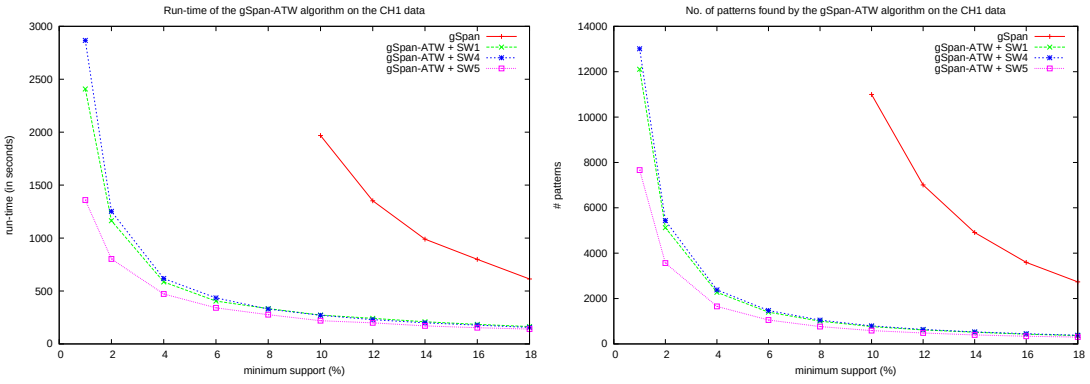


Figure B.18: The performance of gSpan-ATW on the CH1 data

Figure B.19 shows a very similar performance (to that presented in Figure B.18), when gSpan-ATW is applied to CH2.

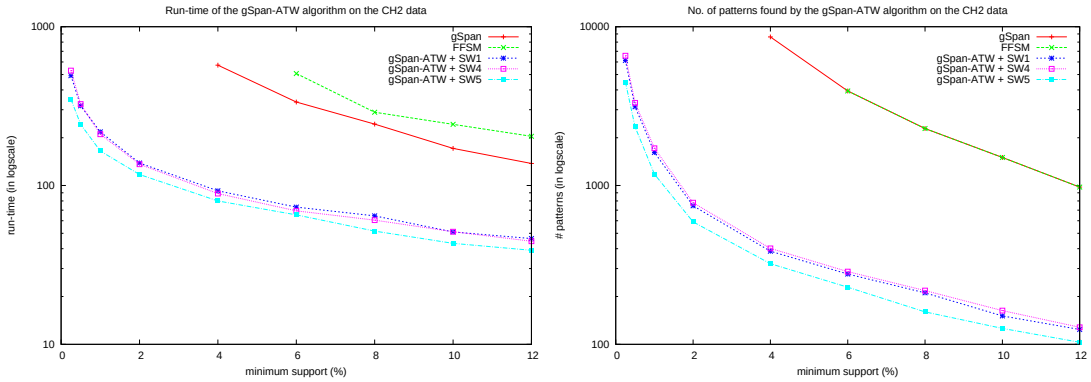


Figure B.19: The performance of gSpan-ATW on the CH2 data

Table B.16: The AUC scores of the classifiers using patterns discovered by gSpan-ATW with SW1 on the CH1 data

Dataset	gSpan					gSpan-ATW + SW1				
	σ(%)	#F	AUC			τ(%)	#F	AUC		
			NBC	SVM	C4.5			NBC	SVM	C4.5
RG1:CH1	16	3596	79.5	69.3	75.5	4	2277	78.9	70.8	75.5
	18	2735	78.8	68.1	75.8	6	1401	77.9	68.8	71.3
	20	2149	78.8	68.4	76.0	8	1002	75.5	67.1	73.6

Table B.17: The AUC scores of the classifiers using patterns discovered by gSpan-ATW with SW4 and SW5 on the CH1 data

Dataset	gSpan-ATW + SW4					gSpan-ATW + SW5				
	τ(%)	#F	AUC			τ(%)	#F	AUC		
			NBC	SVM	C4.5			NBC	SVM	C4.5
RG1:CH1	4	2382	78.5	70.8	76.2	4	1656	79.1	70.2	73.9
	6	1474	78.0	70.9	71.8	6	1053	77.5	70.3	72.1
	8	1051	75.9	68.8	74.0	8	764	76.1	70.2	73.2

B.2.1.2 The RG2 and RG3 data

Table B.18 displays the performance of gSpan-ATW with CW1-N on the RG2 and RG3 data. In comparison with Table 6.5, gSpan-ATW using CW1-N spends less runtime discovering fewer patterns than that using CW1-E on the RG2 data, and the performance of the former is very close to that of the latter.

The accuracy of the classifiers built using patterns discovered by gSpan-ATW with CW1-N on the RG2 and RG3 data, as shown in Table B.19, is similar to that of the classifiers built using patterns discovered by gSpan-ATW with CW1-E (see Table 6.8)

B.2.2 The application of the AW scheme

B.2.2.1 The RG1 data

The performance of the gSpan-AW algorithm on the CH1 data is shown in Figure B.20. As can be seen from the figure, gSpan-AW outperforms gSpan in terms of the runtime

Table B.18: The performance of gSpan-ATW using CW1-N on the RG2 and RG3 data

Dataset	τ(%)	gSpan-ATW + CW1-N	
		runtime (in seconds)	# patterns
RG2:MAM-V80	4	8.436	2126
	6	6.034	247
	8	5.721	81
	10	5.650	80
RG2:MAM-V100	4	19.808	4026
	6	14.729	788
	8	14.718	118
	10	14.667	101
RG3:BS-V500	2	1.644	1341
	4	1.321	607
	6	1.264	497
	8	1.144	456
RG3:BS-V1000	6	0.656	596
	8	0.523	479
	10	0.412	350
	12	0.327	215

Table B.19: The accuracy of the classifiers using patterns discovered by gSpan-ATW with CW1-N on the RG2 and RG3 data

Dataset	gSpan-ATW + CW1-N				
	τ(%)	#F	NBC	SVM	C4.5
RG2:MAM-V80	3	3140	76.5	75.7	68.7
	4	2126	77.4	77.4	71.3
	5	936	69.6	68.7	67.8
RG2:MAM-V100	3	5476	83.5	84.3	70.4
	4	4026	80.0	80.0	70.4
	5	2196	75.7	80.0	65.2
RG3:BS-V500	2	1341	96.5	94.7	87.6
	4	607	95.3	92.9	82.9
	6	497	95.3	93.5	81.8
RG3:BS-V1000	2	911	95.9	92.9	81.2
	4	766	95.3	92.9	81.2
	6	596	93.5	91.8	79.4

cost and the number of patterns identified.

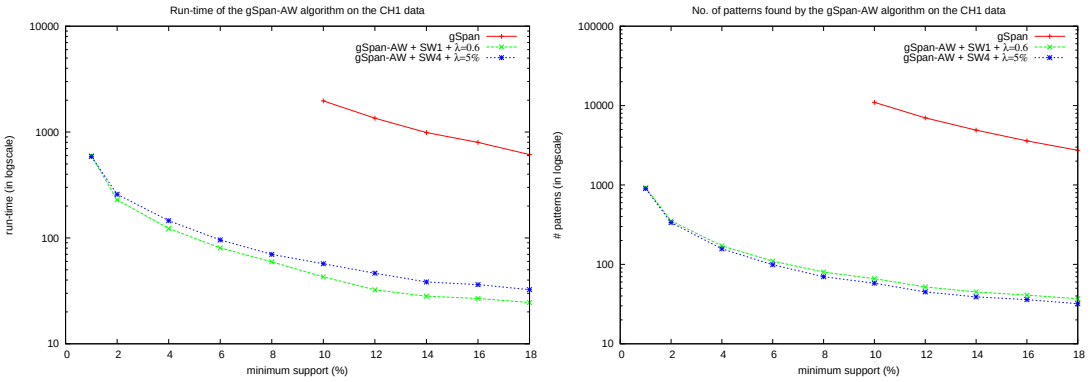


Figure B.20: The performance of gSpan-AW on the CH1 data

Nevertheless, in comparison with the performance of the classifiers built using pat-

Table B.20: The performance of the classifiers using patterns discovered by the standard FSM algorithms on the CH1 data

Dataset	gSpan							
	σ(%)	#F	Accuracy			AUC		
			NBC	SVM	C4.5	NBC	SVM	C4.5
RG1:CH1	16	3596	77.4	80.1	79.5	79.1	69.5	75.5
	18	2735	76.8	79.6	79.2	78.5	67.9	75.8
	20	2149	76.8	79.5	80.2	78.8	68.4	76.0

terns discovered by the standard FSM algorithms as shown in Table B.20, Table B.21 shows that the performance of the classifiers built using patterns discovered by gSpan-AW using SW1 on CH1 is slightly worse than that of the classifiers using patterns discovered by the standard FSM algorithms. In particular for the AUC measure, the NBC classifier built using patterns discovered by gSpan-AW with SW1 achieve considerably lower scores than when using the standard FSM algorithms.

Table B.21: The performance of the classifiers using patterns discovered by gSpan-AW with SW1 on the CH1 data

Dataset	gSpan-AW + SW1								
	λ	τ(%)	#F	Accuracy			AUC		
				NBC	SVM	C4.5	NBC	SVM	C4.5
RG1:CH1	0.6	0.5	2851	73.5	76.2	78.5	68.3	68.7	74.9
		1	922	75.3	76.0	79.4	67.4	67.8	75.5
		2	353	71.1	77.4	79.0	68.8	70.7	74.9

Table B.22: The performance of the classifiers using patterns discovered by gSpan-AW with SW4 on the CH1 data

Dataset	gSpan-AW + SW4								
	λ	τ(%)	#F	Accuracy			AUC		
				NBC	SVM	C4.5	NBC	SVM	C4.5
RG1:CH1	5%	0.5	2822	70.0	76.5	78.2	64.3	67.1	71.1
		1	900	68.9	76.0	76.8	63.3	65.4	69.8
		2	336	68.9	76.4	77.2	63.3	65.6	70.8

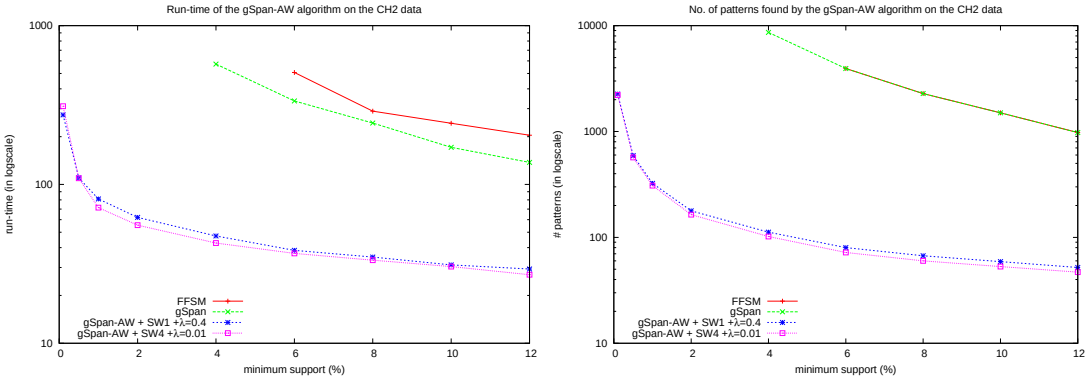
In the case of SW4, the performance of the classifiers built using gSpan-AW, as shown in Table B.22, is even worse than that of the classifiers built using gSpan-AW with SW1.

Table B.23: The performance of gSpan-AW using SW4 on the CH2 data

Dataset	λ	τ(%)	gSpan-AW + SW4	
			runtime (in seconds)	# patterns
RG1:CH2	0.01	4	42.764	102
		6	36.812	72
		8	33.366	60
		10	30.414	53

Figure B.21 shows that gSpan-AW coupled with SW1 or SW4 requires less runtime

and discovers a substantially lower number of patterns than both gSpan and FFSM. Furthermore, the curves representing gSpan-AW coupled with SW1 is very close to that of gSpan coupled with SW4, which indicates that gSpan-AW using either SW1 or SW4 performs very similarly.



(a) (b)

Figure B.21: The performance of gSpan-AW on the CH2 data

B.2.3 The application of the CMW scheme

The experimental results of applying the CMW scheme to undirected graphs are presented in Sub-section B.2.3.1 for the RG1:CH2 data, and Sub-section B.2.3.2 for the RG1:CH1, RG2, and RG3 data.

B.2.3.1 The RG1:CH2 data

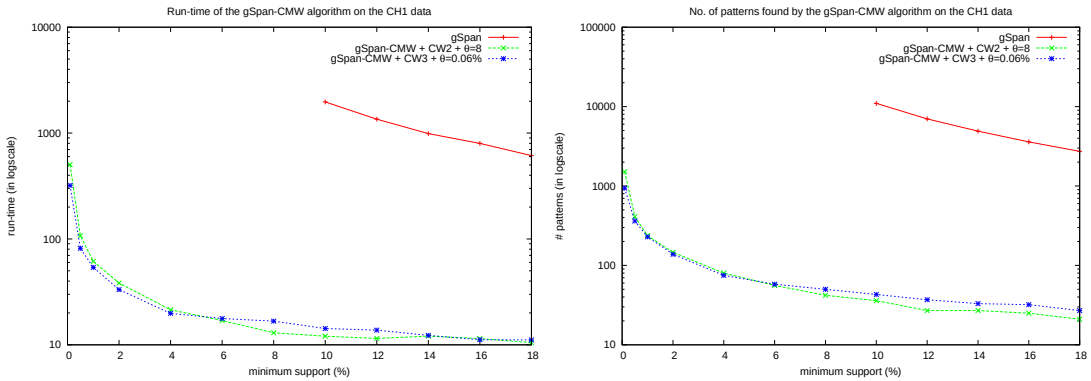
Table B.24: The performance of gSpan-CMW using SW2 on the CH2 data

Dataset	σ(%)	runtime (in seconds)			# patterns	gSpan-CMW + SW2			
		gSpan	FFSM	GASTON		θ	σ(%)	runtime	# patterns
RG1:CH2	4	572.614	740.596	n/a	8624	0.8	4	1220.236	2340
	6	335.605	506.909	n/a	3939		6	577.667	1108
	8	243.612	289.341	n/a	2284		8	366.783	687
	10	171.225	242.826	n/a	1502		10	251.493	465

Table B.25: The performance of gSpan-CMW using SW3 on the CH2 data

Dataset	θ	σ(%)	gSpan-CMW + SW3	
			runtime (in seconds)	# patterns
RG1:CH2	0.8	4	1161.193	2340
		6	589.318	1108
		8	369.980	687
		10	255.337	465

B.2.3.2 The RG1:CH1, RG2, and RG3 data



(a) (b)

Figure B.22: The performance of gSpan-CMW on the CH1 data

Table B.26: The performance of gSpan-CMW using CW3 on the RG1:CH1, RG2 and RG3 data

Dataset	θ	σ(%)	gSpan-CMW + CW3	
			runtime (in seconds)	# patterns
RG1:CH1	0.06%	10	13.121	43
		12	12.238	37
		14	11.043	33
		16	10.672	32
RG2:MAM-V80	0.8%	15	8.593	1500
		17	6.610	623
		19	6.525	353
		21	6.355	323
RG2:MAM-V100	0.8%	15	20.605	3809
		17	17.120	1423
		19	14.738	661
		21	14.435	603
RG3:BS-V500	0.1	8	2.256	2136
		10	1.678	829
		12	1.222	522
		14	1.102	438
RG3:BS-V1000	0.1	6	0.928	656
		8	0.737	506
		10	0.595	366
		12	0.445	219

Table B.27: The AUC scores of the classifiers using patterns discovered by gSpan-CMW with CW2 on the CH1 data

Dataset	gSpan					gSpan-CMW + CW2					
	σ(%)	#F	AUC			θ	σ(%)	#F	AUC		
			NBC	SVM	C4.5				NBC	SVM	C4.5
RG1:CH1	16	3596	79.5	69.5	75.5	8	0.1	1516	75.2	71.5	75.6
	18	2735	78.8	68.1	75.8		0.2	830	75.1	71.8	75.3
	20	2149	78.8	68.4	76.0		0.4	491	75.1	71.9	75.3

Table B.28: The performance of the classifiers using patterns discovered by gSpan-CMW with CW3 on the CH1 data

Dataset	gSpan-CMW + CW3								
	θ	σ(%)	#F	Accuracy			AUC		
				NBC	SVM	C4.5	NBC	SVM	C4.5
RG1:CH1	0.06%	0.1	939	77.0	77.1	78.6	74.0	68.9	75.6
		0.2	636	77.0	77.8	78.6	74.0	69.4	75.7
		0.4	399	77.0	77.8	78.6	74.1	70.0	76.2

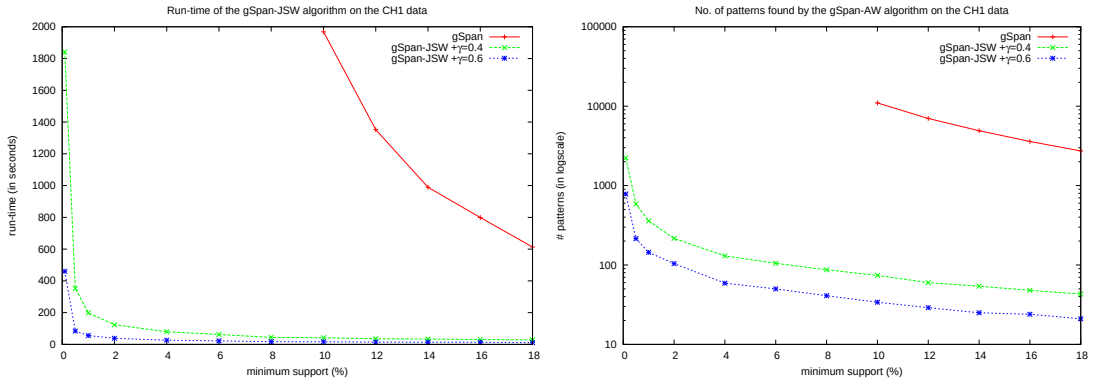
Table B.29: The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW3 on the RG2 and RG3 data

Dataset	gSpan-CMW+ CW3					
	θ	σ(%)	#F	NBC	SVM	C4.5
RG2:MAM-V80	0.8%	15	1500	86.1	76.5	68.7
		17	623	79.1	78.3	76.5
		19	353	80.0	78.3	73.0
RG2:MAM-V100	0.8%	15	3809	90.4	86.1	68.7
		17	1423	81.7	82.6	71.3
		19	661	80.9	82.6	69.6
RG3:BS-V500	0.1	8	2136	95.3	94.7	81.8
		10	829	95.9	94.7	81.8
		12	522	96.5	94.1	81.8
RG3:BS-V1000	0.1	2	2413	95.3	92.9	84.1
		4	993	94.7	92.4	84.1
		6	656	94.1	91.2	85.3

B.2.4 The application of the JSW scheme

B.2.4.1 The RG1 data

Figure B.23 (a) shows that gSpan-JSW with two different γ values, when applied to CH1, runs significantly faster than gSpan. Figure B.23 (b) shows that the number of patterns identified by gSpan-JSW on CH1 is significantly less than those identified by gSpan.



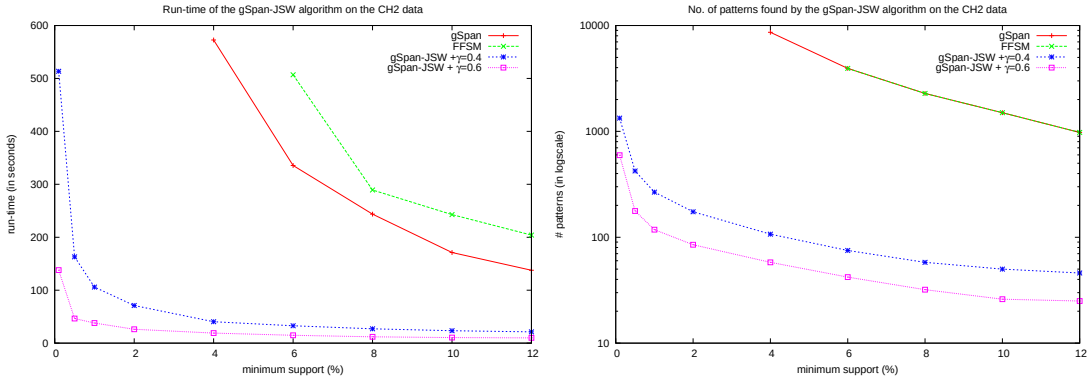
(a) (b)

Figure B.23: The performance of gSpan-JSW on the CH1 data

Table B.30: The AUC scores of the classifiers using patterns discovered by gSpan-JSW on the CH1 data

Dataset	gSpan					gSpan-JSW					
	σ(%)	#F	AUC			γ	σ(%)	#F	AUC		
			NBC	SVM	C4.5				NBC	SVM	C4.5
RG1:CH1	16	3596	79.1	69.5	75.5	0.6	0.1	783	73.3	68.9	76.4
	18	2735	78.5	67.9	75.8		0.2	404	73.3	69.1	76.6
	20	2149	78.8	68.4	76.0		0.4	248	72.7	68.6	75.2

The performance of gSpan-JSW on the CH2 data is presented in Figure B.24. Note that gSpan performs better than the FFSM algorithm, and both gSpan and FFSM run significantly slower, and discover more patterns, than gSpan-JSW using two different γ values. Figure B.24 further indicates that gSpan-JSW, with a large γ value, is more efficient than with a small γ value.



(a) (b)

Figure B.24: The performance of gSpan-JSW on the CH2 data

B.2.5 The application of the UBW scheme

Table B.31: The AUC scores of the classifiers using patterns discovered by gSpan-UBW with SW1 on the CH1 data

Dataset	gSpan					gSpan-UBW + SW1					
	σ(%)	#F	AUC			μ	τ(%)	#F	AUC		
			NBC	SVM	C4.5				NBC	SVM	C4.5
RG1:CH1	16	3596	79.1	69.5	75.5	0.2%	10	171	68.6	67.6	73.2
	18	2735	78.5	67.9	75.8		12	120	68.7	66.5	76.1
	20	2149	78.8	68.4	76.0		14	83	67.3	60.3	72.3

B.3 The Experimental Results for Directed Graphs

Two groups of directed graph data (RG4 and RG5) were used for the evaluation. Some results of using various subgraph weighting schemes on these data are described in the following sub-sections.

Table B.32: The performance of the classifiers using patterns discovered by gSpan-UBW with SW4 on the CH1 data

Dataset	gSpan-UBW + SW4								
	μ	τ(%)	#F	Accuracy			AUC		
				NBC	SVM	C4.5	NBC	SVM	C4.5
RG1:CH1	0.2%	10	164	73.3	76.6	78.8	68.3	66.3	74.6
		12	116	74.5	75.8	77.2	68.7	65.0	74.6
		14	79	72.1	74.9	76.6	67.9	59.9	73.8

B.3.1 The application of the ATW scheme

B.3.1.1 The RG4 data

Table B.33: The accuracy of the classifiers using patterns discovered by gSpan-ATW with CW1-N on the RG4 data

Dataset	gSpan-ATW + CW1-N				
	τ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.2	3319	72.9	72.3	73.1
	0.4	1780	72.9	72.6	73.1
	0.6	1203	72.9	72.6	73.1
RG4:Amazon	0.4	2875	92.4	92.7	91.2
	0.6	2166	92.4	92.7	91.2
	0.8	1653	92.5	93.0	91.2
RG4:Ohsumed	0.4	2646	76.9	78.7	74.7
	0.6	2010	76.9	78.4	74.7
	0.8	1582	77.4	78.6	75.4

B.3.2 The application of the CMW scheme

B.3.2.1 The RG4 data

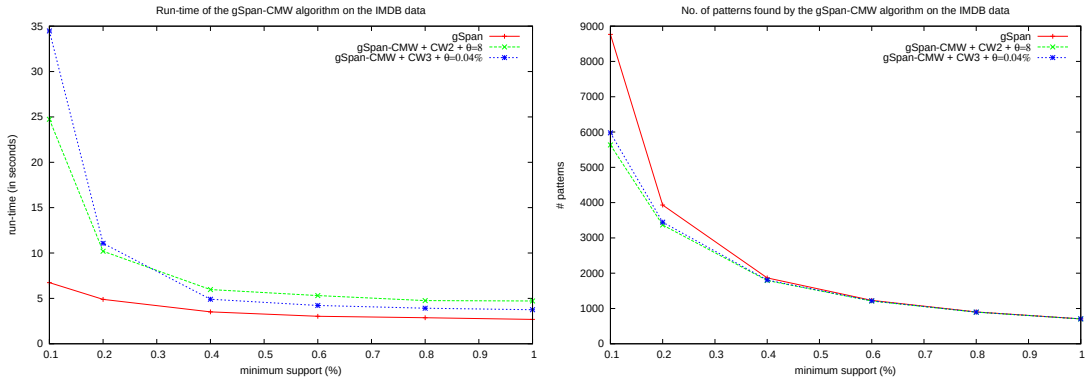


Figure B.25: The performance of gSpan-CMW using CW2 and CW3 on the IMDB data

The performance of gSpan-CMW coupled with CW2 or CW3 on the RG4 data is shown in Figures B.25 and B.26. As can be seen from the figures, gSpan-CMW requires more runtime to discover a smaller number of patterns than gSpan. The reason for this

fact is that the computation of the CW2 or CW3 neutralizes any gain obtained using the CMW scheme.

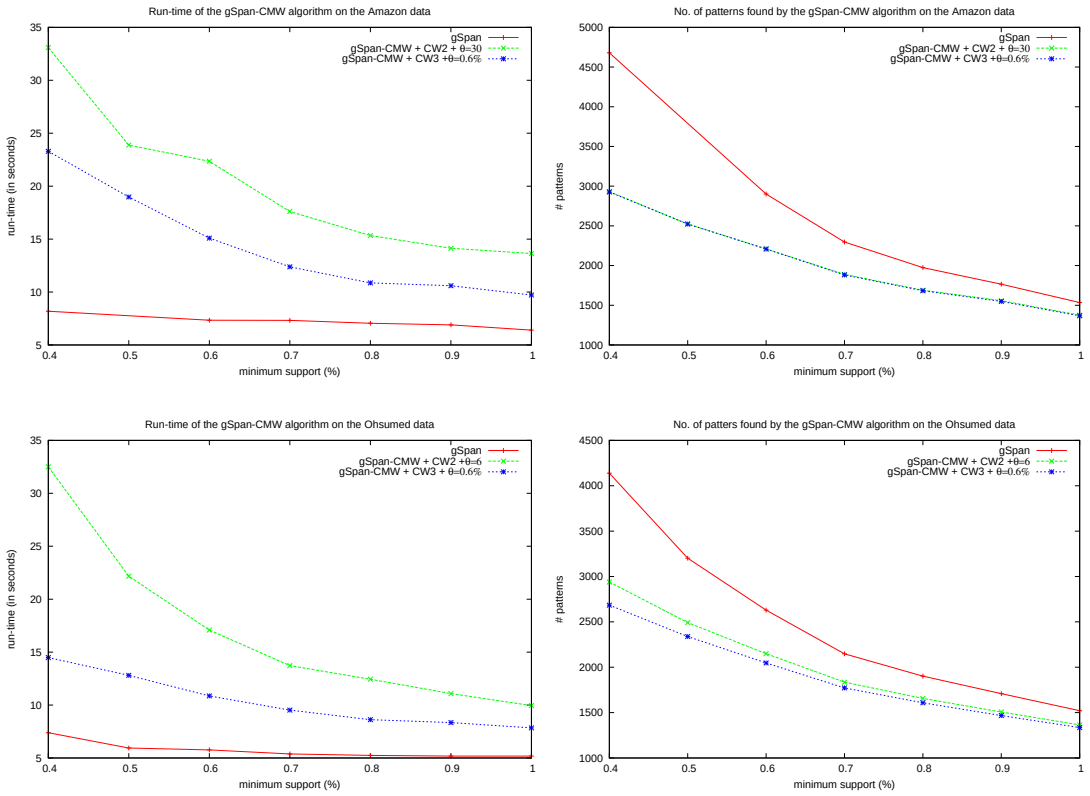


Figure B.26: The performance of gSpan-CMW using CW2 and CW3 on the Amazon & Ohsumed data

The patterns discovered by gSpan-CMW using CW2 or CW3 on the RG4 data were used to construct feature vectors. These feature vectors were then used with respect to a range of classifiers. The results of the classifiers on the RG4 data are displayed in Tables B.34 and B.35. It can be seen from the tables that the classifiers built using patterns discovered by gSpan-CMW achieved higher accuracy results than those built using patterns discovered by gSpan, and the accuracy of the former was achieved using a smaller number of features than that of the latter.

B.3.2.2 The RG5 data

For the Lancashire data, the performance of gSpan-CMW, as shown in Figure B.27 (a), suggests that gSpan-CMW coupled with SW3 runs faster than when coupled with SW2 when the support threshold is over 6% and that the former runs slower than the latter when the support threshold is below 6%. In the case of the Scotland data the performance of gSpan-CMW and gSpan, as shown in Figure B.28, is similar until the support threshold is reduced to below 16%. The advantage offered by gSpan-CMW becomes noticeable when the support threshold is lowered to below 16%. Figure B.28

Table B.34: The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW2 on the RG4 data

Dataset	gSpan					gSpan-CMW + CW2					
	σ(%)	#F	NBC	SVM	C4.5	θ	σ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.1	8768	72.9	71.8	73	8	0.2	3366	72.9	72.1	73.0
	0.2	3932	72.9	72.1	73.0		0.4	1794	74.0	73.0	73.1
	0.4	1866	72.9	72.5	73.1		0.6	1210	74.1	73.2	73.1
RG4:Amazon	0.4	4680	92.4	92.7	91.2	30	0.4	2931	92.4	92.7	91.2
	0.6	2901	92.4	92.7	91.2		0.6	2210	92.4	92.7	91.2
	0.8	1974	92.5	93.0	91.2		0.8	1690	92.5	93.0	91.2
RG4:Ohsumed	0.4	4138	76.9	78.5	74.7	6	0.4	2939	76.9	78.5	74.7
	0.6	2630	76.9	78.2	74.7		0.6	2150	76.9	78.2	74.7
	0.8	1902	76.9	77.8	74.7		0.8	1656	76.9	77.8	74.7

Table B.35: The accuracy of the classifiers using patterns discovered by gSpan-CMW with CW3 on the RG4 data

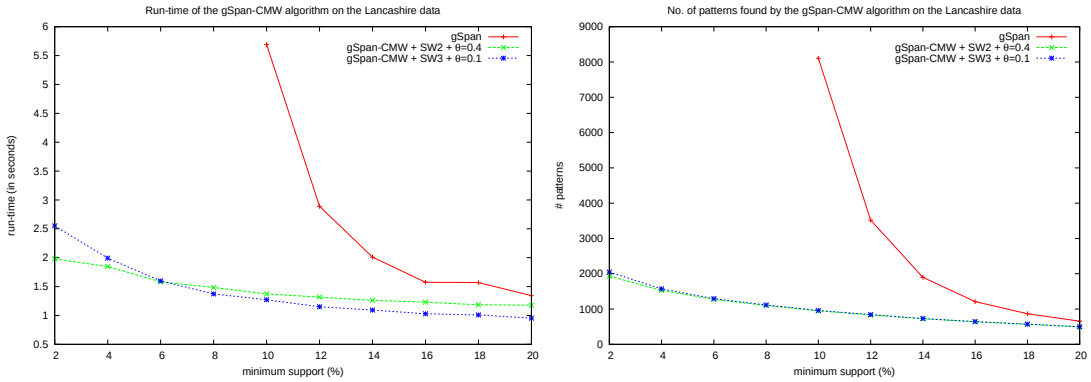
Dataset	gSpan-CMW + CW3					
	θ	σ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.08%	0.2	3366	72.9	72.1	73.0
		0.4	1794	72.9	72.5	73.1
		0.6	1210	72.9	72.5	73.1
RG4:Amazon	0.6%	0.4	2927	92.4	92.7	91.2
		0.6	2208	92.4	92.7	91.2
		0.8	1683	92.4	93.0	91.2
RG4:Ohsumed	0.6%	0.4	2684	76.9	78.5	74.7
		0.6	2047	76.9	78.2	74.7
		0.8	1608	76.9	77.8	74.7

Table B.36: The accuracy of the classifiers using patterns discovered by gSpan-CMW with SW3 on the RG4 data

Dataset	gSpan-CMW + SW3					
	θ	σ(%)	#F	NBC	SVM	C4.5
RG4:IMDB	0.2	0.2	3338	72.9	72.1	72.3
		0.4	1780	72.9	72.6	73.1
		0.6	1203	72.9	72.6	73.1
RG4:Amazon	0.1	0.4	2914	92.4	92.7	91.2
		0.6	2195	92.4	92.7	91.2
		0.8	1673	92.4	93.0	91.2
RG4:Ohsumed	0.2	0.4	2693	76.9	78.5	74.7
		0.6	2035	76.9	78.2	74.7
		0.8	1597	76.9	77.8	74.7

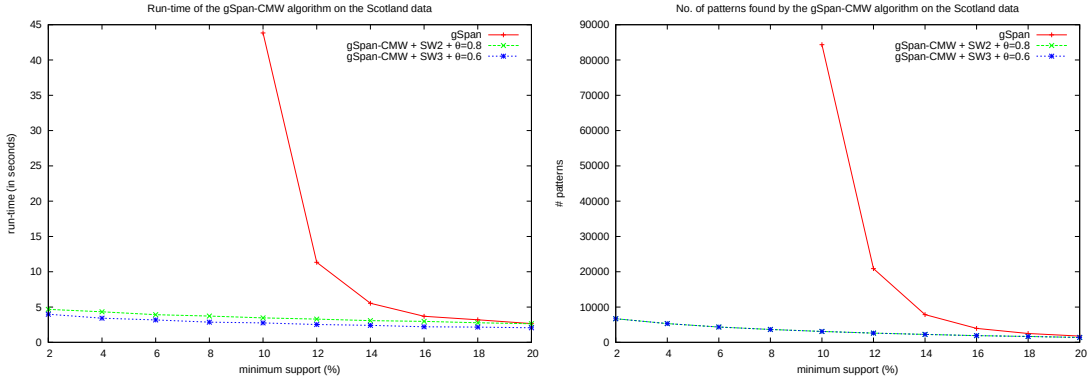
(a) further suggests that gSpan-CMW coupled with SW3 runs a slightly faster than when using SW2.

The performance of gSpan-CMW on the GB data is presented in Figure B.29. In the figure, the benefits of gSpan-CMW start to be evident when the support threshold is lowered to below 20%; the performance of gSpan-CMW is very similar to that of gSpan when the support threshold is above 20%. The reason for this is that the CMW scheme is not very effective at reducing the search space when discovering small sized



(a) (b)

Figure B.27: The performance of gSpan-CMW on the Lancashire data



(a) (b)

Figure B.28: The performance of gSpan-CMW on the Scotland data

patterns. Figure B.29 further demonstrates that gSpan-CMW coupled with SW3 runs faster than when coupled with SW2 when the support threshold is below 20%, but discovers a very similar number of patterns.

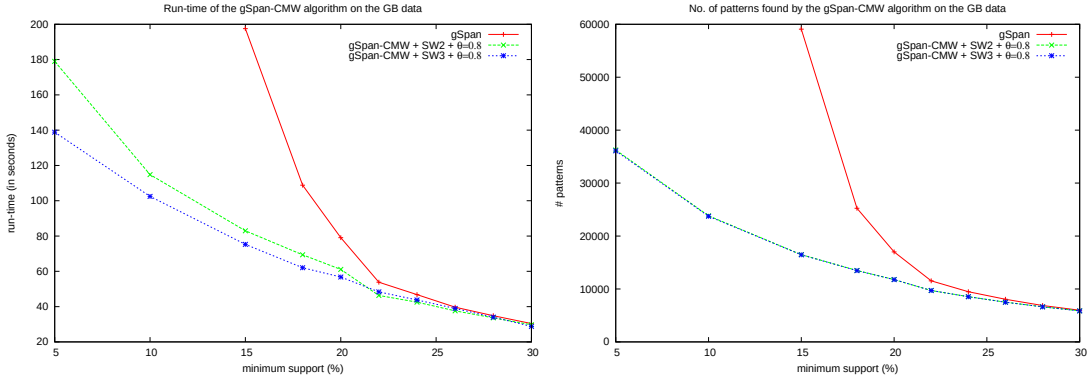
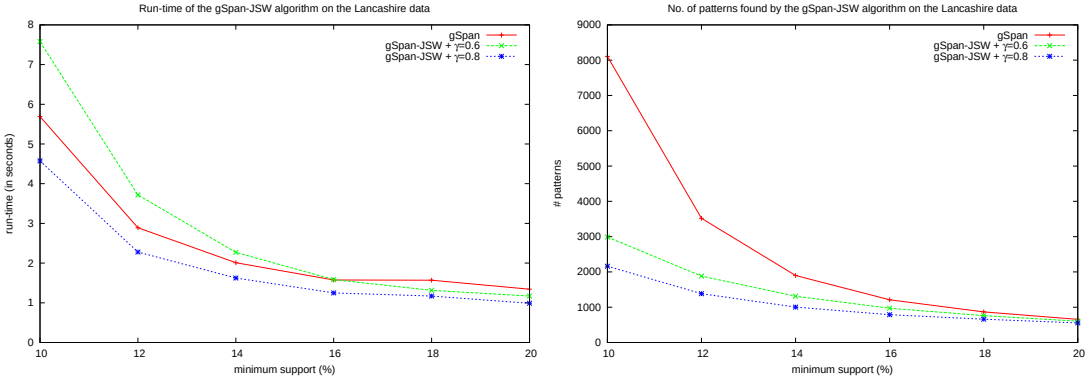


Figure B.29: The performance of gSpan-CMW on the GB data

B.3.3 The application of the JSW scheme

B.3.3.1 The RG5 data

The performance of gSpan-JSW when applied to the Lancashire data set is presented in Figure B.30. From Figure B.30 (a), gSpan-JSW with a smaller γ value runs slower than gSpan when the support threshold is below 16%, while gSpan-JSW with a larger γ value runs consistently faster than gSpan. From Figure B.30 (b) it can be observed that gSpan-JSW (coupled with two different γ values) discovers fewer patterns than gSpan.



(a) (b)

Figure B.30: The performance of gSpan-JSW on the Lancashire data

Figure B.31 shows that the performance of gSpan-JSW when applied to the Scotland data set. In the figure, gSpan-JSW when coupled with two different γ values outperforms gSpan in terms of the runtime and the number of patterns identified. Additionally, gSpan-JSW using a larger γ value runs slightly faster and discovers fewer patterns than when using a smaller γ value. For the largest data set, GB, Figure B.32 exhibits a similar performance to that shown in Figure B.31.

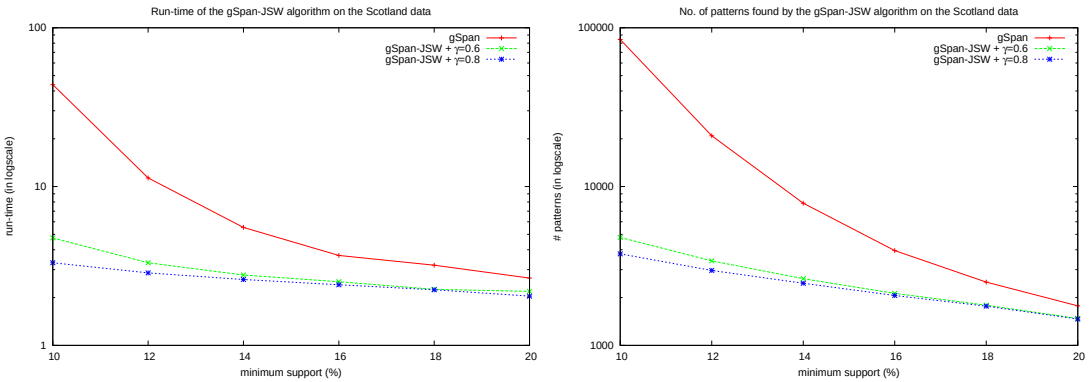


Figure B.31: The performance of gSpan-JSW on the Scotland data

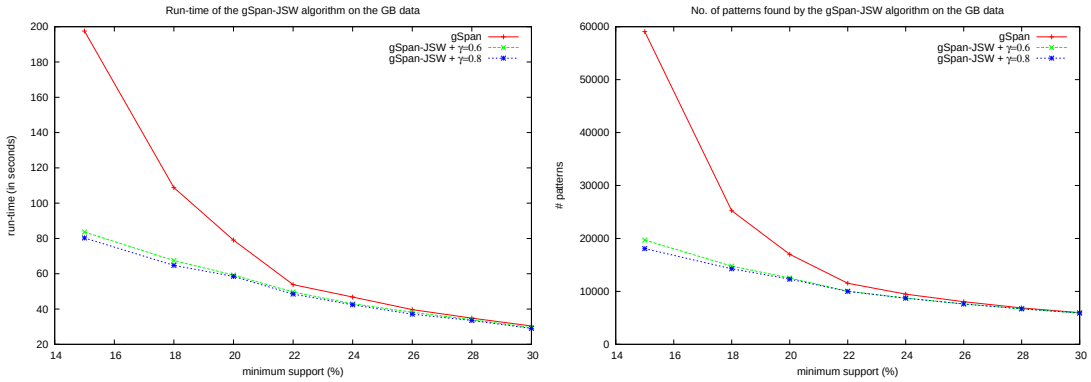


Figure B.32: The performance of gSpan-JSW on the GB data

Appendix C

Published Work

In this final appendix a list of publications to date, including papers submitted for refereeing, from the work described in this thesis is presented:

Journals

1. A. Elsayed, F. Coenen, C. Jiang, M. Garcia-Finana, and V. Sluming. Corpus Callosum MR Image Classification. *Journal of Knowledge Based Systems*, 23(4), pp. 330–336, 2010.

2. C. Jiang, F. Coenen, R. Sanderson, and M. Zito. Text Classification using Graph Mining-Based Feature Extraction. *Journal of Knowledge Based Systems*, 23(4), pp. 302–308, 2010.

3. C. Jiang, F. Coenen, and M. Zito. A Survey of Frequent Subgraph Mining Algorithms. "Accepted by the Knowledge Engineering Review".

Conferences

1. C. Jiang and F. Coenen. Graph-based Image Classification by Weighting Scheme. In *Proceedings of the 2008 SGAI International Conference on Artificial Intelligence (AI'08)*, pp. 63–76, Springer London, 2008.

2. C. Jiang, F. Coenen, R. Sanderson, and M. Zito. Text Classification using Graph Mining based Feature Extraction. In *Proceedings of the 2009 SGAI International Conference on Artificial Intelligence (AI'09)*, pp. 21–34, Springer London, 2009.

3. A. Elsayed, F. Coenen, C. Jiang, M. Garcia-Finana, and V. Sluming. Corpus Callosum MR Image Classification. In *Proceedings of the 2009 SGAI International Conference on Artificial Intelligence (AI'09)*, pp. 333–348, Springer London, 2009 (Winner of the Best Application Paper Prize).

4. C. Jiang, F. Coenen, and M. Zito. Frequent Subgraph Mining on Edge Weighted Graphs. In *Proceedings of the 12th International Conference on Data Warehousing and Knowledge Discovery (DaWak'10)*, pp. 77–88, Springer LNCS, 2010.

5. C. Jiang, F. Coenen and M. Zito. Finding Frequent Subgraphs in Longitudinal Social Network Data Using A Weighted Graph Mining Approach. In *Proceeding of the 6th International Conference on Advanced Data Mining and Applications (ADMA'10)*, Springer LNAI, pp. 405–416, 2010.

6. M.H.A. Hijazi, C. Jiang, F. Coenen, and Y. Zheng. Image Classification for Age-related Macular Degeneration Screening using Hierarchical Image Decomposition and Graph Mining. In *Proceedings of the European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (PKDD'11)*, pp. 65–80, 2011.

Technical Reports

1. A. Elsayed, F. Coenen, C. Jiang, M. Garcia-Finana, and V. Sluming. Segmentation for Medical Image Mining: A Technical Report. Technical Report ULCS-09-016, Department of Computer Science, The University of Liverpool, 2009.

“夫君子之行，静以修身，俭以养德。非淡泊无以明志，非宁静无以致远。夫学须静也，才须学也，非学无以广才，非志无以成学。淫侵则不能励精，险躁则不能治性。年与时驰，意与日去，遂成枯落，多不接世，悲守穷庐，将复何及！”

《诸葛武侯集》