UNIVERSITY OF CALIFORNIA
Santa Barbara

Towards Querying and Mining of Large-Scale Networks

A Dissertation submitted in partial satisfaction
of the requirements for the degree of

Doctor of Philosophy

in

Computer Science

by

Arijit Khan

Committee in Charge:

Professor Xifeng Yan, Chair

Professor Subhash Suri

Professor Divy Agrawal

September 2013

The Dissertation of
Arijit Khan is approved:

Professor Subhash Suri

Professor Divy Agrawal

Professor Xifeng Yan, Committee Chairperson

July 2013

Towards Querying and Mining of Large-Scale Networks

by

Arijit Khan

To my family: Ma, Baba, and Bhai

Acknowledgements

"And, when you want something, all the universe conspires in helping you to achieve it."

Paulo Coelho, in *'The Alchemist'*

During my PhD study, I have been helped, encouraged, and supported by many people whom I would like to acknowledge my deepest gratitude.

My sincere thanks to my adviser Professor Xifeng Yan for his guidance, suggestions, and invaluable encouragement throughout the development of this thesis. His creativity, insightful discussions, and passion for research provided a productive atmosphere for my work. I am very grateful to Professor Divy Agrawal, Professor Amr Abbadi, Professor Ben Zhao, Professor Subhash Suri, and Alessandra Sala for their suggestion and help at various phases of my PhD study. I owe special thanks to Professor Janet Kayfetz and Professor Doug Bradley for helping me in improving my writing and presentation skills.

My thanks to all my collaborators for keeping trust in my capabilities. I would like to express my gratefulness to them: Charu Aggarwal, Shu Tao, Supriyo Chakraborty, Vishwakarma Singh, Shengqi Yang, Nandish Jayaram, Mahesh Gupta, and Professor Chengkai Li. I want to mention four of my collaborators separately. First, Yinghui Wu and Francesco Gullo — I enjoyed discussing several problems with them, and they helped me a lot in improving my scientific writing skill. Second, my sincere and deepest gratitude to my mentors Francesco Bonchi and Aris Gionis for always being my well-wishers. The collaboration started just as a regular summer internship — but, then I learnt a lot about research skills from them. They provided me the freedom to think independently, and helped me to improve myself in many different ways.

Even after the internship, they are always very encouraging, careful, and supportive of me — I hope to continue learning from them, and collaborating with them.

Many thanks to all my Professors and collaborators from my undergraduate years: Professor Subhas Nandy, Professor Debashish Saha, Professor Pradip Das, Professor Lawrence Jenkins, Professor Dilip Basu, Professor Saswata Sannigrahi, Adway Mitra, and Arpan Roy. They will always remain an inspiration who inculcated the quest for higher studies in me.

I am very fortunate to have a close circle of friends from my undergraduate days: Sayan Bhattacharya, Amitangshu, and Shibamouli. Specifically thanks to Rajdeep Sau, Alessandro, Francois, and Daniel Vaquero – who became good friends, and made life at Santa Barbara enjoyable for me.

Last but not least, my family, relatives, and cousins. I would have had no chance to complete my studies without the love, support, and encouragement of my younger brother Surajit, my parents Bina and Rampada: acknowledging them is never enough.

This research was sponsored in part by NSF IIS 0847925, IIS-0905084, IIS 0954125, and by the Army Research Laboratory under cooperative agreements W911NF-09-2-0053. The views and conclusions contained herein are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Army Research Laboratory or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notice herein.

Curriculum Vitæ

Arijit Khan

Education

2008-2013 (expected)	*PhD*, *Computer Science*, University of California, Santa Barbara, CA
	Thesis: Towards Querying and Mining of Large-Scale Networks
2004-2008	*B.E.*, *Computer Science and Engineering*, Jadavpur University, India
	First Class (Honors), ranked 1st among 62 students

Selected Publications

Referred Conference

VLDB 2013	Arijit Khan, Yinghui Wu, Charu Aggarwal, and Xifeng Yan, *NeMa: Fast Graph Search with Label Similarity*, in Proc. of Very Large Data Bases 2013.
SIGMOD 2012	Shengqi Yang, Xifeng Yan, Bo Zong, and Arijit Khan, *Towards Effective Partition Management for Large Graphs*, in Proc. of International Conference on Management of Data 2012.
CIKM 2012	Nan Li, Xifeng Yan, Zhen Wen, and Arijit Khan, *Density Index and Proximity Search in Large Graphs*, in Proc. of ACM Conference on Information and Knowledge Management 2012.
SIGMOD 2011	Arijit Khan, Nan Li, Xifeng Yan, Ziyu Guan, Supriyo Chakraborty and Shu Tao, *Neighborhood Based Fast Graph Search in Large Networks*, in Proc. of International Conference on Management of Data 2011.
SDM 2011	Charu Aggarwal, Arijit Khan, and Xifeng Yan, *On Flow Authority Discovery in Social Networks*, in Proc. of SIAM Conference of Data Mining 2011.

SIGMOD 2010 Arijit Khan, Xifeng Yan, and Kun-Lung Wu, *Towards Proximity Pattern Mining in Large Graphs*, in Proc. of the International Conference on Management of Data 2010.

SAS 2008 Arpan Roy, Adway Mitra, Arijit Khan, Mita Nasipuri, and Debashis Saha, *LSDC: a Lossless Approach to Lifetime Maximization in Wireless Sensor Networks*, in Proc. of Sensors Applications Symposium 2008.

Under Submission

Submitted Arijit Khan, Francesco Bonchi, Aris Gionis, and Francesco Gullo, *RQtree: an Index for Reliability Queries*.

Submitted Nandish Jayaram, Mahesh Gupta, Arijit Khan, Chengkai Li, Xifeng Yan, and Ramez Elmasri, *GQBE: Querying EntityRelationship Graphs by Example Tuples*.

Submitted Charu Aggarawal, Arijit Khan, and Xifeng Yan *GMatrix: A 3-dimensional Synopsis for Massive Graph Streams.*

Tutorial

ICDE 2012 Arijit Khan, Yinghui Wu, and Xifeng Yan, *Emerging Graph Queries In Linked Data*, in Seminar of International Conference in Data Engineering 2012.

Workshop Papers

GDM 2012 Arijit Khan, Vishwakarma Singh, and Jian Wu, *Find Skyline Nodes in Large Networks*, in Proc. of International Workshop on Graph Data Management: Techniques and Applications 2012, co-located with International Conference in Data Engineering 2012 (ICDE 2012).

CloudMan 2012 Arijit Khan, Xifeng Yan, Shu Tao, and Nikos Anerousis, *Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach*, co-located with Network Operations and Management Symposium 2012 (NOMS 2012).

COMSWARE 2008 Arijit Khan and Lawrence Jenkins, *Undersea Wireless Sensor Network for Ocean Pollution Prevention*, in Proc. of Communication Systems Software and Middleware and Workshops, (COMSWARE 2008), IEEE.

Honors and Awards

Received, IBM Ph.D. Fellowship 2012-13.

Received, NSF ICDE 2012 Scholarship.

Received, SDM Student Travel Award 2011.

Received, P1 fellowship 2009-10, Computer Science, University of California, Santa Barbara.

Received, CITRIX GO-TO fellowship 2008-2009, Computer Science, University of California, Santa Barbara.

Winner, Gold Medal from the Department of Computer Science and Engineering, Jadavpur University in 2008.

Winner, Gold Medal by Tata Consultancy Services Ltd. for being the best student of the Department of Computer Science and Engineering, Jadavpur University in 2008.

Winner, Award for Academic Excellence by a Student, 2004 by Telegraph School Awards.

Received, Agarwal Rashtriya Puraskar (Award), 2002 & 2004 presented by Shri Viren J. Shah, then Governor of West Bengal, India.

Received, National Merit Scholarship, India from 2004 to 2008.

Research Experience

Sep. 2008-Now	**Computer Science, UC Santa Barbara**, Santa Barbara, CA *Graduate Research and Teaching Assistant*
Jun. 2012-Sep. 2012	**Yahoo! Research Lab**, Barcelona, Spain *Research Intern*, Mentor: Francesco Bonchi and Aris Gionis.
Jun. 2010-Sep. 2010	**IBM T. J. Watson Research Center**, Hawthorne, New York *Research Intern*, Mentor: Shu Tao, Manager: Nikos Anerousis.
Jun. 2011-Jul. 2011	**Center for Nanotechnology, UC Santa Barbara**, Santa Barbara, CA *INSET Research Mentor*.
Jun. 2007-Sep. 2007	**Electrical Engineering, Indian Institute of Science**, India *Summer Research Fellow*, Mentor: Lawrence Jenkins.

Selected Talks

Jun. 2012	*Towards Querying Large-Scale Heterogeneous Networks*; **Yahoo! Research, Barcelona, Spain**.
Apr. 2012	*Emerging Graph Queries in Linked Data*, Tutorial in the International Conference in Data Engineering 2012, (**ICDE 2012**).
Dec. 2011	*Novel Graph Queries in Large Networks*, **Invited Talk** at **Computer Science, IIT Guwahati, India**.

Jun. 2011	*Neighborhood Based Fast Graph Search in Large Networks*, in the International Conference on Management of Data 2011, (**SIGMOD 2011**).
May 2011	*On Flow Authority Discovery in Social Networks*, in SIAM Conference of Data Mining 2011, (**SDM 2011**).
Jun. 2010	*Towards Proximity Pattern Mining in Large Graphs*, in the International Conference on Management of Data 2010, (**SIGMOD 2010**).
Aug. 2010	*Workload Characterization and Prediction in the Cloud: A Multiple Time Series Approach*, at **IBM T. J. Watson Research Center** in Hawthorne, New York.
Aug. 2007	*Undersea Wireless Sensor Network Protocols for Ocean Pollution Prevention*, at Electrical Engineering, **Indian Institute of Science, India**.

Professional Activities

Reviewer for Journal of Knowledge Based Systems (Elsevier), Journal of Information Systems (Elsevier), IEEE Transactions on Knowledge and Data Engineering, International Conference on Information Systems, Fundamenta Informaticae, and Journal of Zhejiang University Science.

External Reviewer for IEEE Transactions on Neural Networks, VLDB 2014, MLG 2013, SIGMOD 2013, ICDE 2013, WWW 2013, VLDB 2013, ICDM 2012, SIGMOD 2011, SDM 2011, SIGKDD 2010, SDM 2010, SIGMOD 2010, ICDM 2010, ICDM 2009, ICDE 2009.

Student member of the ACM and IEEE.

Student member of the Information Network Academic Research Center (INARC), University of California, Santa Barbara.

Member in General Committee and Best Paper Committee in Graduate Student Workshop, Computer Science, University of California Santa Barbara, 2011 and 2012.

Conferences Attended: SIGMOD 2010, 2011, SDM 2011, and KDD 2011.

Abstract

Towards Querying and Mining of Large-Scale Networks

Arijit Khan

With the advent of the internet, sources of data have increased dramatically, including the World Wide Web, social networks, knowledge graphs, medical and government records. Oftentimes, relations exist among the entities in these data. Therefore, we observe structures in the data, but these structures are implicit, and not as rigid or regular as found in standard database systems. These semi-structured data are usually represented as large networks with labeled nodes and edges. Querying and mining of these linked datasets are essential for a wide range of emerging applications, such as viral marketing, web search, malware detection, image retrieval, and social networks analysis. However, the complex combinations of structure and content, coupled with the massive volume of these data, raise several challenges that require new efforts for smarter and faster graph analysis.

My research interests span the emerging problems in large-scale, heterogeneous, semi-structured data, with a focus on querying and pattern mining in social and information networks using scalable algorithms and machine learning techniques. My research on large-scale graphs could be categorized into two broad directions: (1) querying of large-scale networks, including heterogeneous networks, uncertain and stream graphs, and (2) pattern mining over large graphs. In the domain of querying heterogeneous networks, due to noise and lack of schema, structured methods such as SPARQL — which require an underlying schema to

formulate a query — are often too restrictive. Without knowing the exact structure of the data and the semantics of the entity labels and their relationships, can we still query them and obtain the relevant results? In addition, how do we query uncertain graphs and streams? In the area of graph pattern mining, what graph features one should extract in order to build an accurate and efficient classifier over large networks? From the perspective of advertising and viral marketing, what are the top-k most interesting itemsets and the top-k most influential persons in a social network? In my dissertation, I shall discuss our effective and efficient techniques to solve these emerging problems associated with querying and mining of complex Big-Graphs.

Professor Xifeng Yan
Dissertation Committee Chair

Contents

Acknowledgements **v**

Curriculum Vitæ **vii**

Abstract **xiii**

List of Figures **xviii**

List of Tables **xx**

1 Introduction **1**
1.1 Contributions . . . 2
1.1.1 Graph Querying . . . 3
1.1.2 Graph Pattern Mining . . . 5
1.2 Source Codes and Datasets . . . 6
1.3 Outlines . . . 6

2 Heterogeneous Networks Search **7**
2.1 Introduction . . . 8
2.2 Related Work . . . 13
2.3 Preliminaries . . . 15
2.3.1 Target Graphs, Queries and Matching . . . 16
2.3.2 Subgraph Matching Cost Function . . . 17
2.3.3 Cost Function Properties . . . 20
2.4 Problem Formulation . . . 21
2.5 Query Processing Algorithm . . . 24
2.5.1 Iterative Inference Algorithm . . . 25
2.5.2 Generalized Queries . . . 32
2.6 Indexing and Optimization . . . 34
2.6.1 Candidate Selection . . . 34

2.6.2 Indexing . . . 36
2.6.3 Optimization for Top-k Matching . . . 37
2.7 Experimental Results . . . 38
2.7.1 Experimental Setup . . . 38
2.7.2 Effectiveness and Efficiency . . . 43
2.7.3 Scalability . . . 50
2.7.4 Optimization Techniques . . . 50
2.8 Summary . . . 51

3 Reliability in Uncertain Graphs **53**
3.1 Introduction . . . 54
3.2 Related Work . . . 60
3.3 Problem Statement . . . 61
3.4 The RQ-Tree index: Overview . . . 63
3.5 Query Processing . . . 65
3.5.1 Candidate Generation . . . 65
3.5.2 Verification . . . 78
3.6 Building the RQ-tree Index . . . 82
3.7 Experimental Results . . . 87
3.7.1 Experiments Settings . . . 88
3.7.2 General Performance . . . 93
3.7.3 Insights into the Candidate-Generation Phase . . . 97
3.7.4 Performance with Varying Source-Set Size . . . 98
3.7.5 Scalability . . . 100
3.7.6 Application: Influence Maximization . . . 100
3.8 Summary . . . 103

4 Social Influence Maximization **104**
4.1 Introduction . . . 105
4.2 Related Work . . . 107
4.3 Flow Authority Model for Social Networks . . . 108
4.4 Determining Optimal Information Flow Authorities . . . 113
4.4.1 The BayesTraceback Algorithm . . . 116
4.4.2 Restricting Source and Target Nodes . . . 121
4.5 Experimental Results . . . 122
4.5.1 Data Sets . . . 124
4.5.2 Case Studies . . . 126
4.5.3 Effectiveness Results . . . 128
4.5.4 Efficiency Results . . . 129
4.5.5 Robustness and Scalability with increasing Network Size . . . 133
4.5.6 Targeted Flow Authorities . . . 134
4.6 Summary . . . 135

5 Graph Pattern Mining **136**
5.1 Introduction . . . 137
5.2 Related Work . . . 140
5.3 Preliminaries . . . 142
5.4 Neighbor Association Model . . . 144
5.5 Information Propagation Model . . . 146
5.5.1 Nearest Probabilistic Association . . . 148
5.5.2 Normalized Probabilistic Association . . . 153
5.5.3 Modification for Weighted Graphs . . . 155
5.6 Probabilistic Itemset Mining . . . 155
5.6.1 Exact Mining . . . 156
5.6.2 Approximate Mining . . . 160
5.6.3 Top-k Interesting Patterns . . . 161
5.7 Experimental Results . . . 163
5.7.1 Graph Datasets . . . 163
5.7.2 Effectiveness . . . 165
5.7.3 Efficiency and Scalability . . . 168
5.7.4 Exact vs. Approximate Mining . . . 171
5.7.5 Frequent Subgraph Mining . . . 172
5.8 Summary . . . 173

6 Conclusion and Future Directions **176**
6.1 Concluding Discussion . . . 176
6.2 Future Directions . . . 179

Bibliography **181**

List of Figures

1.1 My Contributions 2

2.1 NeMa: A Query and Its Match (Example 2.1) 9
2.2 Example of False Exact Match in NeMa 21
2.3 Iterative Inference Algorithm NemaInfer 26
2.4 NeMa: Optimal Subgraph Match Finding Algorithm 32
2.5 NeMa: Query Performance 38
2.6 NeMa: Performance against Label Noise 39
2.7 NeMa: Query Performance vs. Noise; IMDB; n_Q: Query Nodes, D_Q: Query Diameter 41
2.8 NeMa: Query Performance vs. Noise; YAGO; n_Q: Query Nodes, D_Q: Query Diameter 41
2.9 NeMa: # Candidates vs. Label Noise 42
2.10 NeMa: # Iterations vs. Label Noise 42
2.11 NeMa: Effectiveness with Unlabeled Query Nodes 43
2.12 NeMa: Performance with Edge Labels (IMDB) 43
2.13 Comparison Results (IMDB): NESS, BLINKS Modified for Approximate Label Match. NESS Results Correspond to its *Filtering* Phase. 45
2.14 NeMa: Scalability 50
2.15 NeMa: Index Performance 51

3.1 Run-Through Example: An Uncertain Graph. 55
3.2 An RQ-tree Index for the Uncertain Graph in Figure 3.1 64
3.3 Reliability Queries: Cumulative Distribution of Arc Probabilities. 90
3.4 RQ-tree Candidate Generation: Results on DBLP, Flickr, Biomine 96
3.5 RQ-tree Candidate Generation: Results on DBLP with Varying Arc Probabilities 96
3.6 Influence Maximization: Last.FM 102
3.7 Influence Maximization: NetHEPT 102

4.1 Determining the Expected Information Spread for a Given Starting Set of Nodes 111

4.2 The RankedReplace Algorithm . . . 114
4.3 The BayesTraceback Algorithm . . . 117
4.4 Influence Maximization: Effectiveness Results (DBLP) . . . 124
4.5 Influence Maximization: Effectiveness Results (Last.FM) . . . 124
4.6 Influence Maximization: Effectiveness Results (Twitter) . . . 124
4.7 Influence Maximization: Efficiency Results (DBLP) . . . 125
4.8 Influence Maximization: Efficiency Results (Last.FM) . . . 125
4.9 Influence Maximization: Efficiency Results (Twitter) . . . 125
4.10 Influence Maximization: Effectiveness vs. Network Size (DBLP) . . . 129
4.11 Influence Maximization: Effectiveness vs. Network Size (Last.FM) . . . 129
4.12 Influence Maximization: Effectiveness vs. Network Size (Twitter) . . . 130
4.13 Influence Maximization: Efficiency vs. Network Size (DBLP) . . . 130
4.14 Influence Maximization: Efficiency vs. Network Size (Last.FM) . . . 130
4.15 Influence Maximization: Efficiency vs. Network Size (Twitter) . . . 131
4.16 Maximum Aggregate Flow for Particular Target Nodes (DBLP) . . . 132
4.17 Running Time to Find Authority Set for Particular Target Nodes (DBLP) . . . 132

5.1 Proximity Pattern $\{a, b, c\}$ . . . 138
5.2 Frequent Itemset vs Proximity Pattern . . . 143
5.3 Overlapping Graph . . . 145
5.4 Consistency: NPA and Frequent Itemset . . . 150
5.5 Support vs Structure Difference . . . 151
5.6 Problem with NPA . . . 152
5.7 FP-Tree for Table 5.2 . . . 158
5.8 pFP applied on Table 5.2 . . . 159
5.9 NmPA Time (DBLP) . . . 169
5.10 NmPA Time vs. # of Labels (DBLP) . . . 170
5.11 Mining Time vs. # of Nodes (DBLP) . . . 170
5.12 Mining Time vs. # of Labels (DBLP) . . . 172

List of Tables

2.1 NeMa vs. Keyword Search and Graph Querying Methods. 11
2.2 NeMa Notations: Target Graphs, Queries and Subgraph Matching 17
2.3 NeMa: Dataset Sizes 39
2.4 NeMa: Query Performance with Varying h (YAGO) 47

3.1 Time Complexity of Reliable Set Computation 58
3.2 Reliability Queries: Dataset Characteristics. 89
3.3 Comparison between RQ-tree and Baselines: Recall. 90
3.4 Comparison between RQ-tree and Baselines: Query-Processing Time (sec). . 91
3.5 RQ-tree: Recall over Various Datasets (Single-Source Queries). 92
3.6 RQ-tree: Query-Processing Time (sec) over Various Datasets (Single-Source). 92
3.7 RQ-tree: Recall w/ Varying Arc Probabilities (DBLP, Single-Source Queries). 92
3.8 RQ-tree: Query Processing Time (sec) w/ Varying Arc Probabilities (DBLP, Single-Source Queries). 93
3.9 RQ-tree: Index Building Time, Index Size, Height of the Tree, and Number of Clusters. 95
3.10 RQ-tree-LB Query-Processing Results on DBLP ($\mu = 5$, $\eta = 0.6$), Varying the Size of the Set of Query Nodes (1^{st} Column) and the Diameter (d) of the Subgraph from which these Nodes were Picked. 99
3.11 Scalability Analysis using Single-Source Queries with $\eta = 0.6$ on the Web-Graph dataset. 99

4.1 Top-20 Results Obtained by Different Influence Maximization Methods 123

5.1 NPA Intermediate Dataset for Figure 5.5 151
5.2 Proximity Patterns: Intermediate Dataset 158
5.3 Top-5 Proximity Patterns (Last.FM) 165
5.4 Proximity Patterns minus Frequent Itemsets (Last.FM) 166
5.5 Top-5 Proximity Patterns (Alerts) 166
5.6 Proximity Patterns minus Frequent Itemsets (Alerts) 167

5.7 Top-5 Proximity Patterns (DBLP) 167
5.8 Proximity Pattern: Runtime (sec) 168
5.9 Proximity Patterns minus Frequent Itemsets using Exact Mining Algorithm (Last.FM) 171
5.10 Proximity Pattern: Runtime Comparison (sec) (Last.FM) 171
5.11 Significant Patterns via LEAP (Last.FM) 172

Chapter 1

Introduction

> *"If you can write, you can code. If you can sketch, you can use a graph database."*
>
> Neo4j

Recent advances in social and information science have shown that linked data pervade our society and the natural world around us [153]. Therefore, graphs have become ubiquitous models to represent complex structures and schema-less data such as Wikipedia, Freebase [64], and various social and information networks. Many of these data are often represented as *heterogeneous* graphs, where the nodes are labeled entities, and the edges represent the relation between two entities. In many novel applications, uncertainty is also inherent in the data due to a variety of reasons, such as noisy measurements [9], inference and prediction models [5,103], or explicit manipulation, e.g., for privacy purposes [26]. In these cases, data can further be represented as an *uncertain* graph, also called *probabilistic* graph, i.e., a graph whose arcs are labeled with a probability of existence. In addition, many graphs such as those defined by the activity on social networks, communication networks, or telephone networks are defined dynamically by

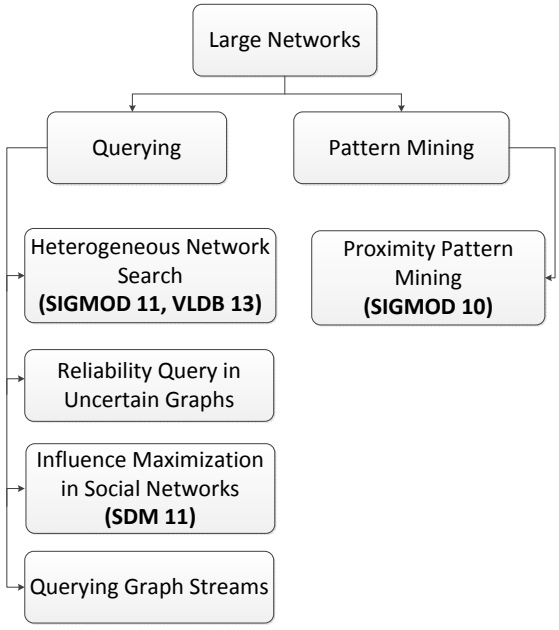


Figure 1.1: My Contributions
Source codes and datasets of our accepted papers are publicly available (Section 1.2)

fast edge *streams* on a massive domain of nodes. Querying and mining such graph data are essential for a wide range of emerging applications including intelligence, predictive analytic, social network analysis, decision and process management.

Most of the existing graph algorithms do not perform well for complex and large networks. My thesis describes the efforts in developing effective and efficient techniques to solve the emerging problems associated with querying and mining of such complex graphs.

1.1 Contributions

My research in the area of large-scale networks can be broadly categorized into two directions — graph querying and pattern mining. Figure 1.1 provides an outline of my works.

1.1.1 Graph Querying

In the broad area of graph querying, I have worked on heterogeneous networks search [86, 95, 96], reliability queries in uncertain graphs [94], influence maximization problem in social networks [7], and querying graph streams [6].

Heterogeneous Networks Search. It is increasingly common to find real-life data represented as networks of labeled, heterogeneous entities. To query these networks, one often needs to identify the matches of a given query graph in a (typically large) network modeled as a target graph. The subgraph isomorphism problem is **NP**-hard [43]. In addition, due to noise and the lack of fixed schema in the target graph, the query graph can substantially differ from its matches in the target graph in both structure and node labels, thus bringing challenges to the graph querying tasks. We propose NeMa (Network Match) [96] — neighborhood structure and label similarity-based fast approximate subgraph matching techniques for querying real-life networks.

Reliability Queries in Uncertain Graphs. Due to noisy measurements, inference errors, and other causes, in many emerging application domains data are represented in the form of uncertain graphs, that is graphs whose arcs are associated with a probability of existence. A fundamental problem on such graphs is to compute the reliable set $RS(S, \eta)$ — the set of all nodes that are reachable from a query set of nodes S with probability no less than a given threshold η. Reliable set computation is a generalization of the source-to-target reliability problem, which is known to be **#P**-complete [148].

In this work [94], we propose RQ-tree, a novel index for efficiently estimating the reliable set, which is based on a hierarchical clustering of the nodes in the graph, and further optimized using balanced minimum cut techniques. Based on RQ-tree, we define a fast filtering-and-verification online query evaluation strategy that relies on a maximum-flow-based candidate-generation phase, followed by a verification phase consisting of either a lower-bounding method or a sampling technique. The first verification method does not return any incorrect node, thus guaranteeing perfect precision, completely avoids sampling, and is more efficient. The second verification method ensures instead better recall.

Influence Maximization in Social Networks. A central characteristic of social networks is that it facilitates rapid dissemination of information between large groups of individuals. This work examines the problem of determination of information *flow representatives* — a small group of authoritative representatives to whom the dissemination of a piece of information leads to the maximum spread. The problem of finding the top-k flow representatives in a social network is **NP**-hard [92]. Therefore, we first design a heuristic RankedReplace algorithm, and then propose a BayesTraceback model in order to approximately find the top-k flow representatives with the use of a fast algorithm [7].

Querying Graph Streams. In many practical settings, the graphs may be drawn on a massive set of nodes, and the edges may arrive rapidly in the form of a graph stream — such as those defined by the activity on social networks, communication networks or telephone networks. It requires a huge space to store all the graph steams over time, and query answering over graph stream are also inefficient since each query might require to process massive streams from the

past. In this work [6], we examine the problem of synopsis construction of massive graph streams. We define the GMatrix structure, which is a 3-dimensional synopsis structure that can summarize massive graphs. A key property of the GMatrix structure is that it retains information about the structural behavior of the underlying graph stream. This ensures that it is possible to use this synopsis structure to answer important structural queries such as finding all connected components of the underlying graphs.

1.1.2 Graph Pattern Mining

We have defined a novel graph pattern, called the proximity pattern [97], and proposed efficient techniques to mine such patterns from large-scale networks.

Proximity Pattern Mining. Mining graph patterns in large networks is critical to a variety of applications such as malware detection and biological module discovery. However, frequent subgraphs are often ineffective to capture association existing in these applications, due to the complexity of isomorphism testing and the inelastic pattern definition. In this work [97], we introduce proximity pattern which is a significant departure from the traditional concept of frequent subgraphs. Defined as a set of labels that co-occur in neighborhoods, proximity pattern blurs the boundary between itemset and structure. It relaxes the rigid structure constraint of frequent subgraphs, while introducing connectivity to frequent itemsets. Therefore, it can benefit from both: efficient mining in itemsets and structure proximity from graphs.

1.2 Source Codes and Datasets

We make the source codes and datasets of our accepted papers publicly available for research purposes only. All our codes are sequential codes with in-memory graph representation using the free edition of LEDA library [107]. Specifically, one may download them as follows.

- proximity pattern [97]: `http://habitus.cs.ucsb.edu/SIGMOD10_Proximity_Pattern.zip`,
- NESS [95]: `http://habitus.cs.ucsb.edu/SIGMOD11_Ness.tar.gz`,
- NeMa [96]: `http://habitus.cs.ucsb.edu/VLDB13_NeMa.tar.gz`, and
- Influence Maximization [7]: `http://habitus.cs.ucsb.edu/Gflow.zip`

The workability and repeatability of our proximity pattern and NESS source codes were verified by the SIGMOD RWE committee [2].

1.3 Outlines

This dissertation is structured as follows. Chapter 2 describes NeMa for the heterogeneous network search problem. Chapter 3 presents the RQ-tree indexing method for efficiently answering reliability queries over uncertain graphs. Chapter 4 considers the influence maximization problem, and our algorithms to find the top-k flow authorities. Chapter 5 describes proximity pattern mining over large-scale networks. Finally, Chapter 6 concludes this thesis.

Chapter 2

Heterogeneous Networks Search

> *"Web search is designed to take any open-ended query and give you links that might have answers. Linking things together based on things that youre interested in is a very hard technical problem. Graph Search is designed to take a precise query and give you an answer, rather than links that might provide the answer."*
>
> Mark Zuckerberg

It is increasingly common to find real-life data represented as networks of labeled, heterogeneous entities. To query these networks, one often needs to identify the matches of a given *query graph* in a (typically large) network modeled as a *target graph*. Due to noise and the lack of fixed schema in the target graph, the query graph can substantially differ from its matches in the target graph in both structure and node labels, thus bringing challenges to the graph querying tasks. In this chapter, we propose NeMa (**Ne**twork **Ma**tch), a neighborhood-based subgraph matching technique for querying real-life networks. (1) To measure the quality of the match, we propose a novel subgraph matching cost metric that aggregates the costs of matching individual nodes, and unifies both structure and node label similarities. (2) Based on the metric, we formu-

late the minimum cost subgraph matching problem. Given a query graph and a target graph, the problem is to identify the (top-k) matches of the query graph with minimum costs in the target graph. We show that the problem is **NP**-hard, and also hard to approximate. (3) We propose a heuristic algorithm for solving the problem based on an inference model. In addition, we propose optimization techniques to improve the efficiency of our method. (4) We empirically verify that NeMa is both effective and efficient compared to the keyword search and various state-of-the-art graph querying techniques.

2.1 Introduction

With the advent of the Internet, sources of data have increased dramatically, including the World-Wide Web, social networks, genome databases, knowledge graphs, medical and government records. Such data are often represented as *graphs*, where nodes are labeled entities and edges represent relations among these entities [72, 171]. Querying and mining of graph data are essential for a wide range of emerging applications [3, 74, 133].

To query these graphs, one often needs to identify the matches of a given *query graph* in a (typically large) *target graph*. Traditional graph querying models are usually defined in terms of *subgraph isomorphism* and its extensions (e.g., edit distance), which identify subgraphs that are exactly or approximately isomorphic to query graphs [138,142,171]. In addition, a wide range of query models and languages are proposed — such as SPARQL and XDD for the RDF and XML data — which require a standard schema of queries and target graphs. Nevertheless, the real-life graphs are *complex* and *noisy*, and often lack standardized schemas [3]. Indeed,

(a) the nodes may be heterogeneous, referring to different entities (e.g., persons, companies, documents) [72]. (b) Node labels in a graph often carry rich semantics, e.g., id, urls, personal information, logs, opinions [74]. (c) Worse still, the semantics of entities and their interconnections in various datasets may be different and unknown to users [3]. In this context, a match may not necessarily be (even approximately) isomorphic to the query graph in terms of label and topological equality. Thus, traditional graph querying techniques are not able to capture good quality matches. Consider the following example over the IMDB movie dataset.

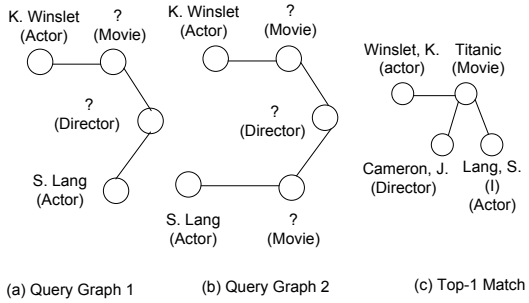


Figure 2.1: NeMa: A Query and Its Match (Example 2.1)

Example 2.1. *A user wants to find a movie of actress 'Kate Winslet' that is directed by the same director who also worked with actor 'Stephen Lang'. Even if the schema and exact entity labels of the target network are not available, the user can still come up with some reasonable graph representation of the query [74, 85], as illustrated in Figure 2.1(a) and 2.1(b). Observe that such graphical representation may not be unique, and there might not be an exact match of the query graph in the dataset. Indeed, the result in Figure 2.1(c) (a star-shaped graph) is by no means similar to the query graphs in Figure 2.1 (a) and (b) (both chain-shaped graphs) under traditional graph similarity definitions. Graph edit distance of the result graph with query*

graphs 1 *and* 2 *are* 4 *and* 6*, respectively. The size of the maximum common subgraph is* 3 *in both cases. Nevertheless, 'Titanic' is the correct answer of the query; and hence, the result graph* should *be considered a good match for both the query graphs using some novel graph similarity metric.*

This motivates us to investigate fast subgraph matching techniques suitable for query answering, which can *relax* rigid structural and label matching constraints of subgraph isomorphism and other traditional graph similarity measures. Our proposed graph similarity metric is based on following observations: (a) if two nodes are close in a query graph, the corresponding nodes in the result graph must also be close. However, (b) there may be some differences in labels of the matched nodes.

While the need for such a graph similarity metric is evident (e.g.,SAGA [142], IsoRank [138]), there is little work on subgraph matching in large networks considering both the criteria. Our previous work, NESS [95] was proposed for subgraph matching that considers the proximity among nodes, but it resorts to strict node label matching. The NESS algorithm is based on a *filtering-and-verification* approach. In the filtering phase, the less promising candidate nodes are pruned iteratively, until no more candidates can be pruned. The output of the filtering phase is a limited number of final candidates for each query node. Then, it verifies all possible graph matches formed by these final candidates, in order to find the top-k graph matches. One can modify NESS to leverage for node label differences. However, this modification reduces the effectiveness of its filtering phase, and results in a large number of final candidates for each query node (See Appendix for an example). Indeed, in our experiments, we find a very low

	NeMa	BLINKS [1]	IsoRank	SAGA	NESS [1]	gStore
Precision (Node)	**0.91**	0.52	0.63	0.75	Filter: 0.17 Filter+Verify: 0.80	0.59
Recall (Node)	**0.91**	0.52	0.63	0.75	Filter: 0.83 Filter+Verify: 0.80	0.59
Precision (Graph)	**0.88**	0.50	0.40	0.69	Filter: 0.39 Filter+Verify: 0.74	0.55
Recall (Graph)	**0.88**	0.50	0.40	0.69	Filter: 0.75 Filter+Verify: 0.74	0.55
Top-1 Match Finding Time (sec)	0.97	1.92	4882.0	15.95	Filter: 0.59 Filter+Verify: 56.16	**0.92**

Table 2.1: NeMa vs. Keyword Search and Graph Querying Methods.

The query graphs were extracted from the IMDB *graph, and later modified by adding* 30% *structural noise and* 50% *label noise. We determined the top-*1 *match for each query graph using various methods, and measured effectiveness at the level of (a) query nodes, and (b) query graphs. At the node level, precision is defined as the ratio of correctly discovered node matches over all discovered node matches, while recall is measured as the ratio of correctly discovered node matches over all correct node matches. Similarly, at the graph level, precision is defined as the ratio of correctly discovered graph matches over all discovered graph matches, and recall is measured as the ratio of correctly discovered graph matches over all correct graph matches. A graph match is considered correct if at least* 70% *of its nodes are matched correctly. Since we consider only the top-*1 *match, precision and recall have the same value. In addition, we also report precision and recall of* NESS *filtering phase. For details about the query graphs, noise, and evaluation metrics, see Section 2.7.*

precision score for NESS, at the end of its filtering phase (Table 2.1). Hence, it becomes quite expensive to determine the top-k graph matches from these large number of final candidates. In

contrast, our proposed NeMa framework employs an inference algorithm that iteratively boosts the score of more promising candidate nodes, considering both label and structural similarity; and thereby directly finds the top-k graph matches.

Contributions. In this work, we propose NeMa, a novel subgraph matching framework for querying heterogeneous networks.

(1) We define the query result as the match of a given query graph in a target graph, in terms of a notion of homomorphism-based subgraph matching. To measure the quality of the matches, we further define a novel subgraph matching cost metric between the query graph and its matches (Section 2.3). In contrast to strict subgraph isomorphism, our proposed metric aggregates the costs of matching individual query nodes, which in turn depends on the cost of matching node labels and their neighborhoods within a certain hops.

(2) Based on the cost metric, we propose the minimum cost subgraph matching problem (Section 2.4), which is to identify the matches of the query graph with minimum costs in the target graph. We show that the problem is **NP**-hard and also hard to approximate.

(3) We propose a heuristic method for the minimum cost subgraph matching problem (Section 2.5). In a nutshell, NeMa converts the underlying graph homomorphism problem into an equivalent inference problem in graphical models [130], and thereby allows us to apply an inference algorithm to heuristically identify the optimal matches. Our method avoids costly subgraph isomorphism and edit distance computations. We further propose indexing and optimization techniques for our method in Section 2.6.

[1] In this chapter, all experimental results with NESS and BLINKS correspond to their modified versions, where we allow two nodes to be matched if their label difference is within a predefined threshold.

(4) We empirically verify the effectiveness and efficiency of NeMa. Our experimental results on real-world networks in Section 2.7 show that NeMa finds better quality results quickly as compared to keyword search (e.g. BLINKS [76]) and various graph querying techniques (e.g.,IsoRank [138], SAGA [142], NESS [95], gStore [171]).

2.2 Related Work

Subgraph Matching. Ullmann's backtracking method [147], VF2 [44], SwiftIndex [137] are used for subgraph isomorphism checking.

The subgraph matching problem identifies all the occurrences of a query graph in the target network. In bioinformatics, exact and approximate *subgraph matching* have been extensively studied, e.g., PathBlast [91], SAGA [142], NetAlign [110], IsoRank [138]. Among them, SAGA is close to ours in terms of problem formulation. However, these algorithms target smaller biological networks. It is difficult to apply them in large heterogeneous networks.

There have been significant studies on inexact subgraph matching in large graphs. Tong et al. [144] proposed the best-effort pattern matching, which aims to maintain the shape of the query. In contrast, we identify the optimal matches in terms of proximity among entities rather than the shape of the query graph. Tian et al. [143] proposed an approximate subgraph matching tool, called TALE, with efficient indexing. Mongiovi et. al. introduced a set-cover-based inexact subgraph matching technique, called SIGMA [119]. Both these techniques use edge misses to measure the quality of a match; and therefore, cannot incorporate the notion of proximity among entities. There are other works on inexact subgraph matching. An in-

complete list (see [66] for surveys) includes homomorphism based subgraph matching [55], belief propagation based net alignment [22], edge-edit-distance based subgraph indexing technique [166], subgraph matching in billion node graphs [140], regular expression based graph pattern matching [21], schema [117] and unbalanced ontology matching [169]. Among them, homomorphism based subgraph matching [55] is close to ours. However, instead of identifying the top-k matches, our work reports all the subgraphs where the query edges can be mapped to paths of a given maximum length and the label differences are within a certain threshold.

There are several works on simulation and bisimulation-based graph pattern matching, e.g., [54, 114], which define subgraph matching as a *relation* among query and target nodes. Compared to them, NeMa, is more strict, since we define subgraph matching as a *function* from query nodes to target nodes.

Label and Concept Propagation. Label propagation has been widely used in semi-supervised learning, e.g., labeling of unlabeled graph nodes [135]. Concept Propagation /Concept Vector, on the other hand, was originally formulated to measure the semantic similarities between terms/concepts in a taxonomy [98]. We note that the spreading activation theory of memory [16] used a similar idea of activation propagation. CP/CV and spreading activation have been effectively used in [41, 95] for approximate structural matching in trees, graphs and also for information retrieval from associated networks [23]. These works consider only strict node label matching. However, subgraph matching without node labels is a harder problem than subgraph matching with node labels [165]. Therefore, instead of strict node label equality, when one al-

lows approximate node label matching (e.g., in our current work), it significantly increases the complexity of the search problem.

Querying Semi-structured Data. Lorel and UnQl are among the preliminary query languages designed for semi-structured data. Both of them model input data as labeled graphs, while permitting users to write queries without detailed knowledge about the schema. Later, an underlying query processing system converts those queries into standard SQL or structural recursion queries, respectively, for retrieving the correct answers. This idea of query rewriting has been explored in the context of both relational and semi-structured data, e.g., [45, 74, 127, 162]. Observe that such query rewriting techniques alleviate users from the complexity of understanding the schema; nevertheless, the underlying query processing system still requires a fixed schema.

In the realm of RDF, SPARQL is widely used as the query processing language. However, writing of a SPARQL query is often too challenging, because it requires the exact knowledge of structure, node labels and types. gStore [171], which is the first study that considers a subgraph matching-based query answering technique in RDF data, allows approximate node label matching, but adheres to strict structural matches. In contrast, our NeMa framework permits both structural and node label mismatches.

Our work is different from the keyword search in graphs [76, 89], as our queries have both *structure* and keywords (node labels).

2.3 Preliminaries

We start with a few definitions.

2.3.1 Target Graphs, Queries and Matching

Target graph. A target graph that represents a heterogeneous network dataset can be defined as a labeled, undirected graph $G = (V, E, L)$, with the node set V, edge set E, and a label function L, where (1) each *target node* $u \in V$ represents an entity in the network, (2) each edge $e \in E$ denotes the relationship between two entities, and (3) L is a function that assigns to each node u a label $L(u)$ from a finite alphabet. In practice, the node labels may represent the attributes of the entities, e.g., name, value, etc.

Query graph. A query graph $Q = (V_Q, E_Q, L_Q)$ is an undirected, labeled graph, with a set of *query nodes* V_Q, a set of query edges E_Q, and a label function L_Q, which assigns to each query node $v \in V_Q$ a label $L_Q(v)$ from a finite alphabet.

We next define the *subgraph matching* of a (connected) query graph in a large target network.

Given a target graph $G = (V, E, L)$ and a query graph $Q = (V_Q, E_Q, L_Q)$, (1) a node $u \in V$ is a *candidate* for a query node $v \in V_Q$ if the difference in their labels (i.e., $L(u)$ and $L_Q(v)$, respectively), determined by a given (polynomial-time computable) *label difference function* Δ_L, is less than or equal to a predefined threshold ϵ. We denote as $\mathbb{M}(v)$ the candidate set of the query node v. (2) a *subgraph matching* is a many-to-one function $\phi : V_Q \rightarrow V$, such that, for each query node $v \in V_Q$, $\phi(v) \in \mathbb{M}(v)$.

Remarks. **(1)** The label difference function Δ_L between two node labels can be defined by a variety of criteria, such as the Jaccard similarity, string edit distance, or more sophisticated

semantic metrics, e.g., ontology similarity [45]. In this work, we use Jaccard similarity measure to determine Δ_L (Section 2.7). **(2)** In contrast to strict one-to-one mapping as in traditional subgraph isomorphism tests, we consider a more general many-to-one subgraph matching function. Indeed, two query nodes may have the same match [55, 133]. **(3)** In practice, the nodes in the target and query graphs may be annotated with types (e.g., Figure 2.1 and [74]), where a query node can only be matched with target nodes having the same type. In such cases, our subgraph matching model can be easily adapted to capture the type constraints by refining candidate sets.

$Q(V_Q, E_Q, L_Q)$	query graph
$G(V, E, L)$	target graph
$\phi : V_Q \rightarrow V$	subgraph matching function
Δ_L	label difference function
$\mathbb{M}(v)$	candidate set of node v
$\mathbb{R}_G(u)$	neighborhood vector of node u
$N_\phi(v, u)$	neighborhood matching cost between v and u
$F_\phi(v, u)$	individual node matching cost between v and u
$C(\phi)$	subgraph matching cost function

Table 2.2: NeMa Notations: Target Graphs, Queries and Subgraph Matching

2.3.2 Subgraph Matching Cost Function

There can be many valid matching functions for a given query graph and a target graph [66]. As stated earlier, our novel graph similarity metric must preserve the proximity among node pairs in the query graph, while the labels of the matched nodes should also be similar. Taking this as our guideline, we introduce the *subgraph matching cost function* in NeMa as a metric to measure the goodness of a matching. The function adds up the costs of matching a query node with its candidate, thereby capturing the difference between labels and

neighborhood structures of the two nodes. We first introduce the notion of a neighborhood vector.

Neighborhood vectorization. Given a node u in the target graph G, we represent the neighborhood of u with a *neighborhood vector* $\mathbb{R}_G(u) = \{\langle u', P_G(u,u')\rangle\}$, where u' is a node within h-hops of u, and $P_G(u,u')$ denotes the *proximity* of u' from u in G.

$$P_G(u,u') = \begin{cases} \alpha^{d(u,u')} & \text{if } d(u,u') \leq h; \\ 0 & \text{otherwise.} \end{cases} \tag{2.1}$$

Here, $d(u,u')$ is the distance between u and u'. The *propagation factor* α is a parameter between 0 and 1; and $h > 0$ is the hop number (effectively, the radius) of the neighborhood for vectorization. The neighborhood vector of node u encodes the proximity information from u to its h-hop neighbors. It often suffices to consider small values of h (e.g., $h = 2$), since the relationship between two entities becomes irrelevant as their social distance increases [30].

Based on neighborhood vectors, we now proceed to model the matching cost of the neighborhoods of a query node and a target node. Let us denote the set of neighboring nodes within h-hops of v as $\mathbb{N}(v)$. Given a matching function ϕ, the neighborhood matching cost between v and $u = \phi(v)$, denoted by $N_\phi(v,u)$, is defined as:

$$N_\phi(v,u) = \frac{\sum_{v' \in \mathbb{N}(v)} \Delta_+ \left(P_Q(v,v'), P_G\left(u, \phi(v')\right)\right)}{\sum_{v' \in \mathbb{N}(v)} P_Q(v,v')} \tag{2.2}$$

where $\Delta_+(x,y)$ is a function defined as

$$\Delta_+(x,y) = \begin{cases} x - y, & \text{if } x > y; \\ 0 & \text{otherwise.} \end{cases} \tag{2.3}$$

Intuitively, $N_\phi(v, u)$ measures the matching cost of the neighborhood vectors of v and u. Note that (i) the user issues a query based on her *vague* notion of how the entities are connected in the target graph. Hence, Δ_+ avoids penalizing the cases when two nodes are closer in the target graph, as compared to their corresponding nodes in the query graph. (ii) We normalize $N_\phi(v, u)$ over the neighborhood of v that incurs more cost when same number of node misses occurs in a smaller neighborhood.

Recall that we assume the existence of the label difference function $0 \leq \Delta_L \leq 1$. Now, the individual node matching cost for matching function ϕ is defined as a linear combination of the label difference function and the neighborhood matching cost function.

$$F_\phi(v, u) = \lambda \cdot \Delta_L \left(L_Q(v), L(u)\right) + (1 - \lambda) \cdot N_\phi(v, u), \tag{2.4}$$

where $u = \phi(v)$.

This node matching cost combines *both* label matching cost and neighborhood matching cost via a parameter $0 < \lambda < 1$, whose optimal value lies between $0.3 \sim 0.5$ empirically (Section 2.7).

We are now ready to define our subgraph matching cost function. Given a matching ϕ from the query nodes $v \in V_Q$ to target nodes $\phi(v) \in V$, the subgraph matching cost function is defined as:

$$C(\phi) = \sum_{v \in V_Q} F_\phi(v, \phi(v)) \tag{2.5}$$

Intuitively, $C(\phi)$ is the matching cost of ϕ between the query graph Q and the target graph G, and the problem is to find a matching function ϕ that minimizes $C(\phi)$. Note that,

assuming Δ_L is non-negative, $F_\phi(v, \phi(v))$ and therefore, $C(\phi)$ are both non-negative, so the minimum value that $C(\phi)$ can take is 0.

2.3.3 Cost Function Properties

The following properties of our subgraph matching cost function illustrates its connection with subgraph isomorphism.

Property 2.1. *If the query graph Q is subgraph isomorphic (in terms of structure and node labels equality) to the target graph G, then there exists a minimum cost matching function ϕ with $C(\phi) = 0$.*

Property 2.1 ensures that all the matching functions ϕ, which identifies exact (isomorphic) matches for Q, must have cost 0. However, a match ϕ of Q, where $C(\phi) = 0$, may not necessarily be isomorphic to Q. We refer to such matches as *false exact matches*.

Example 2.1. Consider a query graph Q, a target graph G (Figure 2.2), and a subgraph matching function ϕ, where $\phi(v_1)$=u_1, $\phi(v_2)$= $\phi(v_4)$=u_2, and $\phi(v_3)$=u_3. Assuming $h = 1$ and $\alpha = 0.5$, the neighborhood vectors in Q are: $\mathbb{R}_Q(v_1)$=$\{\langle v_2, 0.5\rangle, \langle v_3, 0.5\rangle\}$, $\mathbb{R}_Q(v_2)$=$\{\langle v_1, 0.5\rangle\}$, $\mathbb{R}_Q(v_3)$=$\{\langle v_1, 0.5\rangle, \langle v_4, 0.5\rangle\}$, and $\mathbb{R}_Q(v_4)$= $\{\langle v_3, 0.5\rangle\}$. Similarly, we have the following neighborhood vectors in G: $\mathbb{R}_G(u_1) = \{\langle u_2, 0.5\rangle, \langle u_3, 0.5\rangle\}$, $\mathbb{R}_G(u_2)$ =$\{\langle u_1, 0.5\rangle, \langle u_3, 0.5\rangle\}$, and $\mathbb{R}_G(u_3) = \{\langle u_1, 0.5\rangle, \langle u_2, 0.5\rangle\}$. Therefore, the individual node matching costs F_ϕ is 0 for all $v \in V_Q$, and the subgraph matching cost $C(\phi)$ is 0. Observe that the match identified by ϕ is not isomorphic to Q.

However, if the matching function ϕ is *one-to-one*, the following property shows that the false exact matches can be avoided,.

Property 2.2. *If the match identified by ϕ is not isomorphic to the query graph Q, and ϕ is a one-to-one function, then $C(\phi) > 0$.*

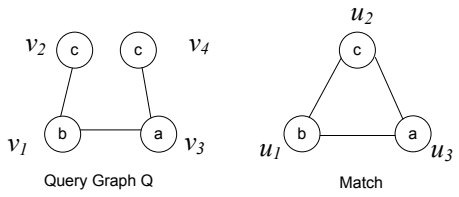


Figure 2.2: Example of False Exact Match in NeMa

Proof Since Q is connected and ϕ is a one-to-one function, if the match identified by ϕ is not isomorphic to Q, one of the following must hold. (1) There exists some node $v \in V_Q$, s.t., $\Delta_L\big(L_Q(v),\, L(\phi(v))\big) > 0$. Then, $C(\phi) > 0$, assuming $\lambda \neq 0$ in Eq. 2.4. (2) There exists an edge (v, v') in E_Q; but the corresponding edge (u, u') is not in graph G. $\phi(v) = u$ and $\phi(v') = u'$. This implies $P_Q(v, v') = \alpha$, but $P_G(u, u') < \alpha$, which in turn implies $N_\phi(v, u) > 0$. Assuming $\lambda \neq 1$ in Eq. 2.4, we get $C(\phi) > 0$. □

2.4 Problem Formulation

The subgraph matching cost function favors matches with low matching costs. Based on the matching cost function, we introduce the minimum cost subgraph matching problem as follows.

Problem 2.1. Minimum Cost Subgraph Matching. *Given a target graph G, a query graph Q, and the label noise threshold ϵ, find the minimum cost matching ϕ,*

$$\operatorname*{argmin}_{\phi} C(\phi), \tag{2.6}$$

$$\textit{s.t.} \qquad \Delta_L\left(L_Q(v), L(u)\right) \leq \epsilon, \forall v \in V_Q, u = \phi(v) \tag{2.7}$$

Intuitively, instead of checking subgraph isomorphism, our problem formulation identifies the optimal match by minimizing node label differences as well as node pair distances. The identified matches serve as answers to the query graph.

The problem is, however, nontrivial. The following theorem shows that the decision version of the problem is intractable, even when the subgraph matching function ϕ is not injective.

Theorem 2.1. *Given a target network G, a query graph Q, it is* **NP***-complete to determine whether there exists a match ϕ with* NeMa *subgraph matching cost $C(\phi) = 0$.*

Proof The problem is **NP**, since there is a nondeterministic algorithm which guesses a matching function ϕ, and verifies whether its cost $C(\phi) = 0$, in polynomial time. We prove the **NP**-hardness by reduction from the graph homomorphism problem, which is **NP**-complete [43]. A homomorphism from a graph Q' to a graph G' (both unlabeled) is a function that preserves node adjacency (i.e., each edge in Q' is mapped to an edge in G'). Given an instance of the graph homomorphism problem, we construct an instance of the minimum cost subgraph matching problem, where all nodes in the target graph G and query graph Q have identical labels. We

also assume, *w.l.o.g.*, that the depth of vectorization $h = 1$. One may verify that if there exists a homomorphism ϕ' from Q' to G', then there exists a corresponding matching ϕ from Q to G, s.t. $C(\phi) = 0$. Conversely, if ϕ' is not a homomorphic matching, then there exists an edge (v, v') in E_Q, but the corresponding edge $(\phi(v), \phi(v'))$ is not in G. Hence, $C(\phi) > 0$ ($\lambda \neq 1$ in Eq. 2.4). Therefore, there exists a matching function ϕ from Q to G, where $C(\phi) = 0$, if and only if there is a homomorphic matching ϕ' from Q' to G'. This completes the proof. □

One may want to find a polynomial time approximation algorithm. However, the problem is also hard to approximate.

Theorem 2.2. *The minimum cost subgraph matching is APX-hard.*

Proof We show that this optimization problem is APX-hard by performing a reduction (f, g) from the Maximum Graph Homomorphism (MGH) problem without self loops, which is APX-hard [126]. An MGH problem identifies a matching which maximizes the number of edges of the query graph Q that can be mapped to edges of the target graph G (both unlabeled). Given an instance I of MGH, we construct an instance I' of the minimum cost subgraph matching problem, where all nodes in the target network G and query graph Q have identical labels. Let n_q and e_q be the total number of nodes and edges, respectively, in Q. *w.l.o.g.*, assume the depth of vectorization $h = 1$, and the proportionality constant $\lambda = 1 - \frac{1}{n_q}$. We denote by OPT$(I)$ the value of the optimal solution of problem instance I, and VAL(I, x) the value of a feasible solution x of the problem instance I. Assume OPT$(I) = e_o$ and VAL$(I, x) = e$ for some feasible solution x of instance I. Clearly, $e_o \geq 1$. Hence, (1) OPT$(I') < 1 \leq e_o =$ OPT(I). Also, given some feasible solution y of instance I', one may verify that |OPT$(I) -$

VAL $(I, g(y))\,| = e_o - e$, and $|\text{OPT}(I') - \text{VAL}(I', y)| \geq \frac{e_o - e}{2n_q e_q}$. Therefore, (2) $|\text{OPT}(I) - \text{VAL}\,(I, g(y))\,| \leq 2n_q e_q |\text{OPT}(I') - \text{VAL}(I', y)|$. Thus, there exists a reduction (f, g) from MGH to the minimum cost subgraph matching problem, and the theorem follows. □

2.5 Query Processing Algorithm

In this section, we propose a heuristic solution to identify the minimum cost matchings. We start by introducing the max-sum inference problem in graphical models [130], and show how our graph homomorphism problem underlying the NeMa framework is equivalent to an inference problem in graphical models.

Max-Sum Inference. In graphical models, the joint probability distribution function $p(X)$ of a set of variables $X = \{x_1, x_2, \ldots, x_M\}$ can be expressed as a product of the form $p(X) = \prod_i f_i(X_i)$, where each $X_i \subseteq X$. Alternatively, $\log p(X) = \sum_i \log f_i(X_i)$. The *Max-Sum* inference problem is to find the values of the variables $x_1, x_2, \ldots, x_M$ that result in maximum $p(X)$. In other words, we would like to maximize $\log p(X)$ that can be decomposed as the sum of several functions of the form $\log f_i(X_i)$, each of which depends only on a subset of the original variables.

The objective of the max-sum inference problem is similar to that of the minimum cost subgraph matching problem, which is to *minimize* the overall subgraph matching cost $C(\phi)$. Recall that (1) $C(\phi)$ is an aggregation of the individual node matching costs $F_\phi(v, \phi(v))$ of all query nodes v, and (2) the individual node matching cost of a query node v depends only on the matches of v and its neighbors in $\mathbb{N}(v)$. In light of this, we propose an *iterative inference* algo-

rithm similar to the loopy belief propagation algorithm [130], used for inferencing in graphical models.

2.5.1 Iterative Inference Algorithm

In this section, we introduce our inference algorithm, denoted as NemaInfer and illustrated in Figure 2.3.

Overview. Given a query graph Q and a target graph G, NemaInfer first computes the candidate set for each query node using the node label similarity function Δ_L (line 1). Next, it initializes an *inference cost* $U_0(v, u)$ by assigning it to the minimum possible value of individual node matching costs $F_\phi(v, u)$, over all possible matching functions ϕ, s.t., $\phi(v) = u$ (line 2-3). It then *iteratively* computes an *inference cost* for each query node v and its candidates, and selects the *optimal match* of v as its candidate u with the minimum inference cost. NemaInfer keeps track of the optimal matches for each query node. The procedure repeats until it reaches a fixpoint, where the optimal matches for more than a threshold number of query nodes remain identical in two successive iterations (lines 4-12). Finally, NemaInfer refines the matches of each query node and its neighborhood that it "memorizes" via a memoization technique, and obtains the best match (line 13). The constructed subgraph match Φ is then returned (line 14).

We next introduce several procedures of NemaInfer in detail.

Inference cost and optimal match (lines 3-12). The algorithm NemaInfer improves the quality of the matching in each iteration, based on the notion of an *inference cost* and the *optimal match*.

Algorithm NemaInfer

Input: Target graph $G(V, E, L)$, Query Graph $Q(V_Q, E_Q, L_Q)$.

Output: Minimum cost matching of Q in G.

1. **for each** node $v \in V_Q$ **do** compute $\mathbb{M}(v)$;
2. $i := 0$; flag := true;
3. Initiate iterative inferencing with Eq. 2.8;
4. **while** flag **do**
5. $i := i + 1$;
6. **for each** $v \in V_Q$ **do**
7. **for each** $u \in \mathbb{M}(v)$ **do**
8. compute $U_i(v, u)$ with Theorem 2.3;
9. keep track of the current matches of neighbors $v' \in \mathbb{N}(v)$;
10. compute optimal match $O_i(v)$ using Eq. 2.10;
11. **if** more than a threshold number of query nodes v satisfy $O_i(v) = O_{i-1}(v)$ **then**
12. flag := false;
13. construct Φ for all $v \in V_Q$ (with Eq. 2.11, 2.12).
14. **return** Φ;

Figure 2.3: Iterative Inference Algorithm NemaInfer

Inference cost. At each iteration i of NemaInfer, the inference cost $U_i(v, u)$ for each $v \in V_Q$ and $u \in \mathbb{M}(v)$ is defined as follows.

$$U_0(v,u) = \min_{\{\phi:\phi(v)=u\}} F_\phi(v,u) \tag{2.8}$$

$$U_i(v,u) = \min_{\{\phi:\phi(v)=u\}} \left[F_\phi(v,u) + \sum_{v' \in \mathbb{N}(v)} U_{i-1}\left(v', u'\right)\right] \tag{2.9}$$

We assume $i > 0$, and $u' = \phi(v')$ in Equation 2.9. Intuitively, the inference cost is the minimum sum of the individual node matching cost $F_\phi(v,u)$ and the previous iteration's inference costs $U_{i-1}\big(v', \phi(v')\big)$ for all neighbors v' of v, over all possible matching functions ϕ, with the constraint $\phi(v) = u$.

Note that although we consider the minimization over all possible matching functions ϕ, s.t., $\phi(v) = u$, in Equation 2.9, it only depends on the matches of the neighboring nodes in $\mathbb{N}(v)$. As discussed later, inference costs can be computed in polynomial time.

Optimal match. In every iteration, we also define the *optimal match* of each query node. The optimal match of a query node v at iteration i, denoted by $O_i(v)$, is defined as follows.

$$O_i(v) = \operatorname*{argmin}_{u \in \mathbb{M}(v)} U_i(v,u); \quad i \geq 0 \tag{2.10}$$

Example 4.1. We illustrate the idea of one iteration of NemaInfer using Figure 2.4. Assume we have already determined the candidate matches $\mathbb{M}(v)$ for every query node v using the label similarity function Δ_L. For example, $\mathbb{M}(v_2) = \{u_2, u_5, u_9\}$ and $\mathbb{M}(v_4) = \{u_7, u_{10}\}$ in Figure 2.4. Also, consider $h = 1$. At $i = 0$, $U_0(v_2, u_5) = U_0(v_2, u_9) = 0$. Therefore, we can not distinguish between u_5 and u_9 in the initialization round, as which one is a better match

of v_2. However, observe that $U_0(v_4, u_{10}) < U_0(v_4, u_7)$. u_{10} is a neighbor of u_9, while u_7 a neighbor of u_5. Thus, it not only influences the optimal match $O_0(v_4)$ of v_4 at iteration $i = 0$, but it also makes $U_1(v_2, u_9) < U_1(v_2, u_5)$ at iteration $i = 1$, via Eq. 2.9. Hence, we improve the matches in each iteration and proceed towards the minimum cost (heuristic) subgraph match.

Invariant. The algorithm NemaInfer posses the following invariant in each of its iteration, which illustrates the connection between the inference cost and the subgraph matching cost (Section 2.3).

Invariant 2.1. *If there exists a matching function ϕ from the nodes of Q to the nodes of G, such that, $C(\phi) = 0$, then $U_i(v, \phi(v)) = 0$ for all $v \in V_Q$ and $i \geq 0$.*

However, the converse is not always true. In fact, based on the properties of the loopy belief propagation algorithm, there is no guarantee that our algorithm will converge for *all* query nodes after a certain number of iterations. Therefore, we terminate the procedure when more than a threshold number of query nodes v satisfy the condition $O_i(v) = O_{i-1}(v)$. We empirically verified in Section 2.7 that our method usually requires about 2 to 3 iterations to terminate — around 95% of query nodes converge using *IMDB* dataset, and also performs well in real-life networks.

Matching refinement (line 13). The optimal match of each query node at the final iteration might not correspond to the subgraph matching function with the minimum (heuristic) aggregate cost [130]. This can happen if there are multiple graph matching functions that result in the minimum cost graph matches. Therefore, we need to refine the optimal node matches from the

final round of NemaInfer to identify one such minimum cost subgraph matching function, say Φ. We refer to the matches of the query nodes corresponding to Φ as the *most probable matches*. To find these most probable matches, the standard *memoization* technique can be used after the termination of our iterative inference algorithm. First, a query node, say v, is selected randomly, and its most probable match, denoted by $\Phi(v) \in \mathbb{M}(v)$, is determined as follows:

$$\Phi(v) = \underset{u \in \mathbb{M}(v)}{\operatorname{argmin}} U_{i'}(v, u) \tag{2.11}$$

In Eq. 2.11, $i = i'$ denotes the final iteration. For the remaining nodes, the most probable matches are determined by memoizing recursively, i.e., we keep track of the matches of the neighboring nodes that give rise to the most probable match of the current node. For example, the most probable matches $\Phi(v')$ of all $v' \in \mathbb{N}(v)$ are obtained using the most probable match of v as follows.

$$\begin{aligned} \phi_p &= \underset{\{\phi : \phi(v) = \Phi(v)\}}{\operatorname{argmin}} \left[F_\phi(v, \Phi(v)) + \sum_{v' \in \mathbb{N}(v)} U_{i'-1}\left(v', \phi(v')\right) \right] \\ \Phi(v') &= \phi_p(v') \end{aligned} \tag{2.12}$$

The aforementioned memoization technique is performed until the most probable matches of all query nodes are computed.

Computation of Inference Costs. A straightforward approach to determine the inference cost $U_i(v, u)$ for a query node v and its candidate u (Eq. 2.9) considers all possible combination of matches for all nodes $v' \in \mathbb{N}(v)$, which has exponential time complexity and might be very

expensive. In this section, we propose a technique to compute the inference cost in *polynomial time*.

Partial inference cost. To evaluate the inference cost $U_i(v, u)$ for a query node v and its candidate u at iteration i of the algorithm NemaInfer, we compute a *partial inference cost* for each node $v' \in \mathbb{N}(v)$, which is denoted by $W_i(v, u, v')$, and defined below.

$$W_i(v, u, v') = \min_{\{\phi:\phi(v)=u\}} \big[\beta(v) \cdot \Delta_+ \left(P_Q(v, v'), P_G\left(u, \phi(v')\right)\right) + U_{i-1}\left(v', \phi(v')\right)\big] \tag{2.13}$$

Here, $\beta(v) = [\sum_{v' \in \mathbb{N}(v)} P_Q(v, v')]^{-1}$. To compute $W_i(v, u, v')$, we only need to find the minimum value in Eq. 2.13 over the candidates in $\mathbb{M}(v')$. Hence, the partial inference cost $W_i(v, u, v')$ can be computed in polynomial time, for each triplet v, u, v', where $u \in \mathbb{M}(v)$ and $v' \in \mathbb{N}(v)$. Next, we show the relation between the partial inference cost and the inference cost.

Theorem 2.3. *The inference cost $U_i(v, u)$ is computable in polynomial time via the following formula:*

$$U_i(v, u) = \Delta_L\big(L_Q(v), L(u)\big) + \sum_{v' \in \mathbb{N}(v)} W_i(v, u, v') \tag{2.14}$$

Proof

$$
\begin{aligned}
& U_i(v,u) \\
&= \min_{\{\phi:\phi(v)=u\}} \underbrace{\Big[\Delta_L\big(L_Q(v),L(u)\big)+\beta(v)\cdot \sum_{v'\in\mathbb{N}(v)} \Delta_+(P_Q(v,v'),P_G(u,\phi(v')))}_{F_\phi(v,u)} \\
&\qquad\qquad + \sum_{v'\in\mathbb{N}(v)} U_{i-1}(v',\phi(v'))\Big] \\
&= \min_{\{\phi:\phi(v)=u\}} \sum_{v'\in\mathbb{N}(v)} \Big[\beta(v)\cdot\Delta_+(P_Q(v,v'),P_G(u,\phi(v')))+U_{i-1}(v',\phi(v'))\Big] \\
&\qquad\qquad + \Delta_L\big(L_Q(v),L(u)\big) \\
&= \sum_{v'\in\mathbb{N}(v)} \underbrace{\min_{\{\phi:\phi(v)=u\}} \Big[\beta(v)\cdot\Delta_+(P_Q(v,v'),P_G(u,\phi(v')))+U_{i-1}(v',\phi(v'))\Big]}_{W_i(v,u,v')} \\
&\qquad\qquad + \Delta_L\big(L_Q(v),L(u)\big) \\
&= \Delta_L\left(L_Q(v),L(u)\right) + \sum_{v'\in\mathbb{N}(v)} W_i(v,u,v')
\end{aligned}
$$

Hence, the theorem. □

It follows from Theorem 2.3 that the inference cost $U_i(v,u)$ of nodes v and u can be efficiently computed in polynomial time, by (1) determining the partial inference cost $W_i(v,u,v')$ for each $v' \in \mathbb{N}(v)$, and (2) aggregating these partial inference costs with $\Delta_L\ \big(L_Q(v),L(u)\big)$. The aforementioned technique also keeps track of which matches of the neighboring query nodes give rise to the most probable match of the current query node. This information is required during matching refinement.

Time complexity. We analyze the time complexity of the algorithm NemaInfer. Let us denote the number of nodes in the target graph G and the query graph Q as $|V|$ and $|V_Q|$, respectively.

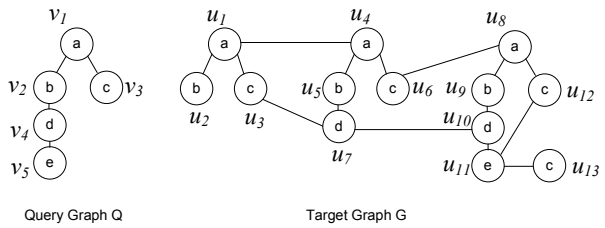


Figure 2.4: NeMa: Optimal Subgraph Match Finding Algorithm

(1) It requires $O(|V_Q| \cdot |V|)$ time to identify $\mathbb{M}(v)$ for each query node $v \in V_Q$ (line 1). (2) We denote the maximum number of candidates per query node as m_Q, and the maximum number of h-hop neighbors of each query node as d_Q. The computation of the optimal match $O_t(v)$ per query node v has time complexity $O(m_Q^2 \cdot d_Q)$ following Theorem 2.3 (line 10). Therefore, the time required for each iteration of NemaInfer is $O(|V_Q| \cdot m_Q^2 \cdot d_Q)$. If there are total I iterations, the overall time complexity is given by $O(|V_Q| \cdot |V| + I \cdot |V_Q| \cdot m_Q^2 \cdot d_Q)$. Observe that $|V_Q|$, I, $|d_Q|$ and m_Q are typically small. Indeed, as verified in our experiments (Section 2.7), I is typically less than 4 and m_Q is 35, for query graphs with 5 nodes and real life graphs containing 12M nodes.

2.5.2 Generalized Queries

In this section, we extend NemaInfer for three generalized cases, namely, *Top-k matches*, *unlabeled queries*, and *labeled edges*.

Top-k Matches. In many applications, the query graph is not subgraph isomorphic to the target network; and hence, we are interested in identifying the top-k matches rather than only the best match. Given the target network G and the query graph Q, the *top-k subgraph matching* problem is to identify the top-k matches for a *selected query node* $v \in V_Q$.

The algorithm NemaInfer can be readily adapted for this problem. (1) The algorithm computes the inference costs and terminates at line 12. (2) We identify the top-k most probable matches of v (Eq. 2.11). (3) For each of these top-k most probable matches of v, we apply the recursive memoizing technique (Eq. 2.12) to determine the corresponding most probable matches for other query nodes.

Matching Query with Unlabeled Nodes. A query graph may have nodes with unknown labels, e.g., query graphs constructed from RDF queries. NeMa can be adapted to evaluate such queries. First, we identify all the nodes from the target network that can be matched with some labeled query node based on label similarity. Next, we find the subgraph induced by all those matched nodes from the target network along with their neighbors within h-hops. All nodes in this subgraph are considered as the candidates for the unlabeled query nodes. The algorithm NemaInfer is then invoked to identify the matches. In addition, if the unlabeled query nodes contain type information, the candidate sets can further be refined.

NeMa **with Edge Labels.** The NeMa cost function can be adapted to consider the edge labels. Specifically, we concatenate the edge labels along the shortest path between a pair of nodes, and then update the neighborhood matching cost (Equation 2.2) as follows.

$$N_\phi(v,u) = \frac{\sum\limits_{v' \in \mathbb{N}(v)} \left[\Delta_+\left(P_Q(v,v'), P_G\left(u,u'\right)\right) + \Delta_L(s(v,v'), s(u,u'))\right]}{\sum\limits_{v' \in \mathbb{N}(v)} \left[P_Q(v,v') + 1\right]}$$

Here, $u = \phi(v)$, $u' = \phi(v')$, and $s(v, v')$ concatenates the edge labels along the shortest path between v and v'. Since, we consider the edge labels along the shortest path between a pair of nodes, the asymptotical time complexity of NeMaInfer remains the same.

2.6 Indexing and Optimization

In this section, we discuss indexing and optimization techniques to improve the efficiency of our network matching algorithm.

2.6.1 Candidate Selection

The candidate set of a query node is defined in terms of the label similarity function (see Section 2.3), which may include candidate nodes that do not match the query node due to neighborhood mismatch. We introduce optimization techniques to efficiently filter such candidate nodes as much as possible, and thereby improving the performance of our inference algorithm NemaInfer. We first introduce the notion of *isolated candidates*.

Isolated Candidates. Given a query node v and its candidate set $\mathbb{M}(v)$, a node $u \in \mathbb{M}(v)$ is an *isolated candidate* of v, if

$$\{u' : u' \in \mathbb{M}(v'), v' \in \mathbb{N}(v)\} \cap \{u'' : u'' \in \mathbb{N}(u)\} = \emptyset \tag{2.15}$$

Intuitively, the node u is an isolated candidate of a query node v if none of the candidates within h-hop neighbors of v are in the h-hop neighborhood of u; otherwise, it is a non-isolated candidate of v. Thus, an isolated candidate u of v can not be matched with v.

To efficiently find the non-isolated candidates, we propose an optimization problem, based on *verification cost* and *candidate cover*.

Verification Cost. The verification cost associated with a query node v is defined as the time complexity to verify all nodes in its candidate set $\mathbb{M}(v)$, whether they are non-isolated candidates. Note that the complexity of verifying whether some node $u \in \mathbb{M}(v)$ is a non-isolated candidate is $O\big(|\mathbb{N}(u)| + \sum_{v' \in \mathbb{N}(v)} |\mathbb{M}\ (v')|\big)$.

Candidate Cover. There exists dependencies between two non-isolated candidates: if u is a non-isolated candidate of v, then there must exist a node $u' \in \mathbb{N}(u)$, such that, $u' \in \mathbb{M}(v')$ for some $v' \in \mathbb{N}(v)$. Clearly, u' is a non-isolated candidate of v'. If we verified all candidates $\{u' : u' \in \mathbb{M}(v'), v' \in \mathbb{N}(v)\}$, there is no need to verify the candidates in $\mathbb{M}(v)$ again. Thus, one may reduce redundant verifications using a notion of candidate cover.

We define *candidate cover* $\mathbb{C}(Q)$ as a set of query nodes v, such that, for all $v' \in V_Q$, either $v' \in \mathbb{C}(Q)$, or $v' \in \mathbb{N}(v)$.

$$\mathbb{C}(Q) = \{v : \forall v'(v' \in \mathbb{C}(Q) \vee v' \in \mathbb{N}(v))\} \tag{2.16}$$

All non-isolated candidate nodes can be identified by verifying *only* the query nodes in $\mathbb{C}(Q)$. We define the verification cost of a candidate cover as the sum of the verification costs of its constituent query nodes. Next, we introduce the candidate cover problem.

Problem 2.2. Candidate Cover. *Given a query graph Q, find the candidate cover with the minimum verification cost.*

The following result shows that the candidate cover problem is intractable, but approximable within a factor 2 in polynomial time.

Theorem 2.4. *The candidate cover problem is (1)* **NP***-hard, and (2)* 2*-approximable.*

Proof We show that this problem is **NP**-hard by reduction from **NP**-complete weighted minimum vertex cover problem [43]. Given a decision version of the weighted minimum vertex cover problem, we construct an instance of the candidate cover problem, where the vertex weights are considered as the corresponding verification costs. Assuming $h = 1$, the minimum weighted vertex cover will be our candidate cover. One can apply linear programming to solve this problem with 2-approximation [43]. □

2.6.2 Indexing

We introduce indexing technique to improve the efficiency of the inference algorithm. (1) During the off-line indexing phase, it computes the neighborhood vectors $\mathbb{R}(u)$ for all nodes u in the target network G, and stores the vectors in the index. (2) During the on-line network matching technique, if u is selected as a candidate of v, it applies Eq. 2.15 to verify whether u is an isolated candidate of v. If so, u is eliminated from the candidate set of v.

Our index structure has space and time complexity $O(nd_G^h)$, where $|V| = n$, $d_G =$ average node degree in G, and $h =$depth of vectorization. For NeMa with edge labels (Section 2.5.2), the asymptotical time and space complexity of indexing remains the same, since we consider edge labels along the shortest paths.

Dynamic maintenance of the index. Our indexing methods can efficiently accommodate dynamic updates in the target network. If a node u (and the edges attached to it) is added or deleted, only the indexes of u's h-hop neighbors need to be updated. If a single edge (u, u') is added or deleted, only the $h-1$ hop neighbors of both u and u' are updated, thus reducing the redundant computation.

2.6.3 Optimization for Top-k Matching

The inference algorithm NemaInfer can be adapted to identify the top-k matches. For small values of k, it is possible to prune candidate nodes by setting a cost threshold. The cost threshold ϵ_c is initially set to a small value ϵ_0. If $U_i(v,u) > \epsilon_c$ for some $u \in \mathbb{M}(v)$ at iteration i of the inference algorithm, then u is eliminated from the candidate set $\mathbb{M}(v)$ for the subsequent iterations. After termination, if the top-k matches cannot be identified, we increase ϵ_c by a small value, and repeat the steps above. The correctness of this method is ensured by Theorem 2.5.

Theorem 2.5. *If $U_i(v,u) > \epsilon_c$ at the i-th round of inference algorithm, then for all $j > i$, $U_j(v,u) > \epsilon_c$ at the j-th round of inference algorithm.*

Proof It follows directly from Eq. 2.9 and the fact that $U_i(v,u) \geq 0$ for all $i \geq 0$, over all pairs v, u, where $u \in \mathbb{M}(v)$. □

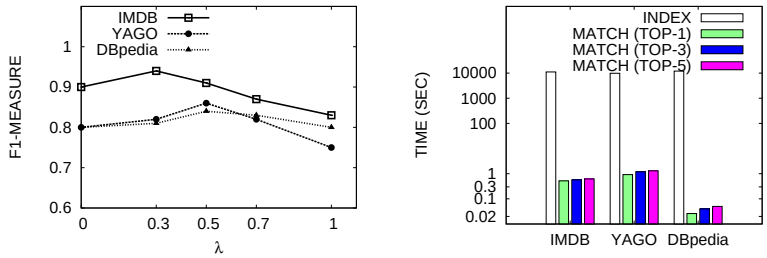
Hence, we can eliminate u from $\mathbb{M}(v)$, whenever $U_i(v,u) > \epsilon_c$ occurs for the first time at some iteration i of NemaInfer.

2.7 Experimental Results

We present three sets of empirical results over three real-life datasets to evaluate (1) the effectiveness and efficiency (Section 2.7.2), (2) scalability (Section 2.7.3), and (3) optimization techniques (Section 2.7.4) underlying the NeMa framework.

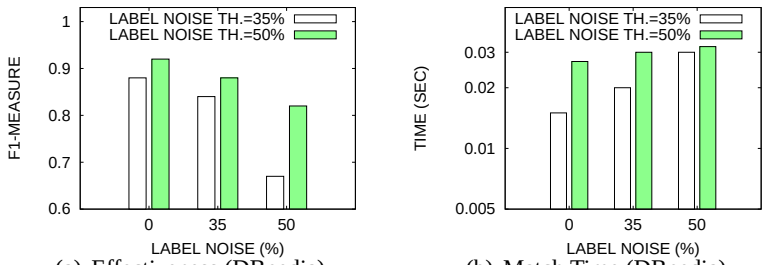
2.7.1 Experimental Setup

Graph Data Sets. We used the following three real life datasets, each represents a target graph. (1) ***IMDB* Network.** [83] The *Internet Movie Database (IMDB)* consists of the entities of movies, TV series, actors, directors, producers, among others, as well as their relationships. (2) ***YAGO* Entity Relationship Graph.** [157] *YAGO* is a knowledge base with information harvested from the Wikipedia, WordNet and GeoNames. It contains about 20 million RDF triples. (3) ***DBpedia* Knowledge Base.** [48] *DBpedia* extracts information from the Wikipedia. We considered 22 million RDF triples from *DBpedia* article categories, infobox properties, and person data.



(a) Effectiveness (b) Efficiency

Figure 2.5: NeMa: Query Performance



(a) Effectiveness (DBpedia) (b) Match Time (DBpedia)

Figure 2.6: NeMa: Performance against Label Noise

Dataset	# Nodes	# Edges
IMDB	2,932,657	11,040,263
YAGO	12,811,149	18,282,215
DBpedia	5,177,018	20,835,327

Table 2.3: NeMa: Dataset Sizes

The nodes in *YAGO* and *DBpedia* are annotated with labels, while the nodes in *IMDB* are annotated with both types and labels. Hence, we used the type information associated with the nodes, in addition to their labels, while querying the *IMDB* network.

Query graphs. We generated the query graphs by extracting subgraphs from the target graphs, and then introduced *noise* to each query graph. Specifically, the query generation was controlled by:

- node number and diameter, denoted by $|V_Q|$ and D_Q, respectively, where the *query diameter* is the maximum distance between any two nodes in the query graph Q.
- *structural noise*, the ratio of the number of edge updates (random insertion and deletion of edges) in Q to the number of edges in the extracted subgraph; and

- *label noise*, measured by the Jaccard similarity between the labels of nodes in the extracted subgraph and their updated counterparts in Q, where the updated labels were obtained by inserting randomly generated words to the query node labels.

We used Jaccard similarity as the label similarity measure. Specifically, given a query node v and a target node u, the label difference $\Delta_L\left(L_Q(v), L(u)\right)$ is defined as $1 - \frac{|w_v \cap w_u|}{|w_v \cup w_u|}$, where w_v and w_u are the set of words in their label $L_Q(v)$ and $L(u)$, respectively. Recall that we allowed some noise in the node labels by varying the label matching cost threshold ϵ in our matching algorithm (Problem Statement 2.1). A node u in the target network is considered a candidate to match with a query node v if their label difference $\Delta_L\left(L_Q(v), L(u)\right)$ is less than the predefined cost threshold ϵ, referred to as the *label noise threshold*.

Evaluation metrics. Since the query graphs were extracted from target graphs, one already has the correct node matches. Now, the effectiveness of NeMa is measured as follows. *Precision* (P) is the ratio of the correctly discovered node matches over all discovered node matches. *Recall* (R) is the ratio of the correctly discovered node matches over all correct node matches. *F1-Measure* combines the results of precision and recall, i.e.,

$$F1 = \frac{2}{(1/R + 1/P)} \tag{2.17}$$

We considered the top-1 match to evaluate precision, recall, and F1-measure. Thus, we obtained the same values for them. However, precision and recall will be useful while analyzing NESS.

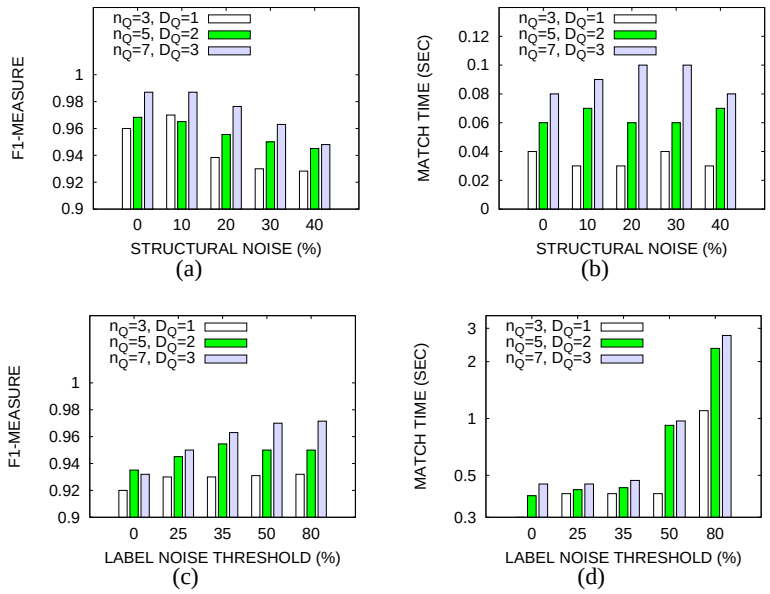


Figure 2.7: NeMa: Query Performance vs. Noise; IMDB; n_Q: Query Nodes, D_Q: Query Diameter

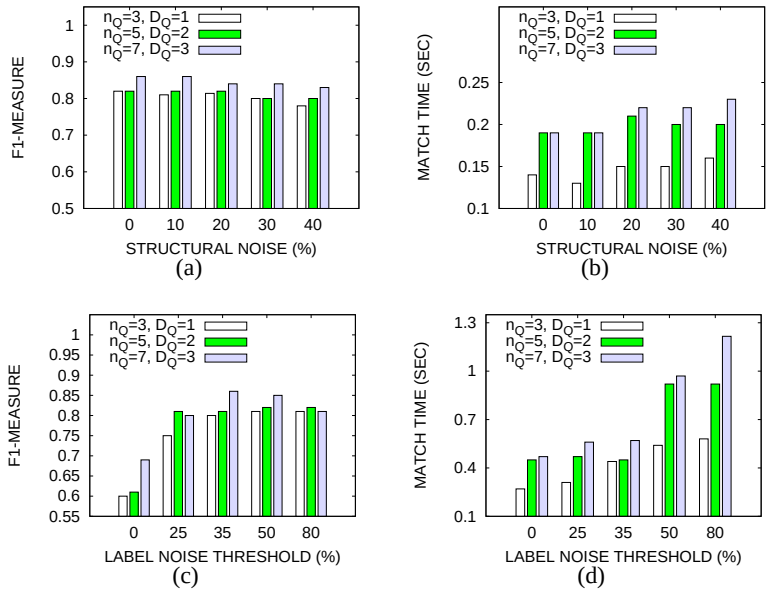
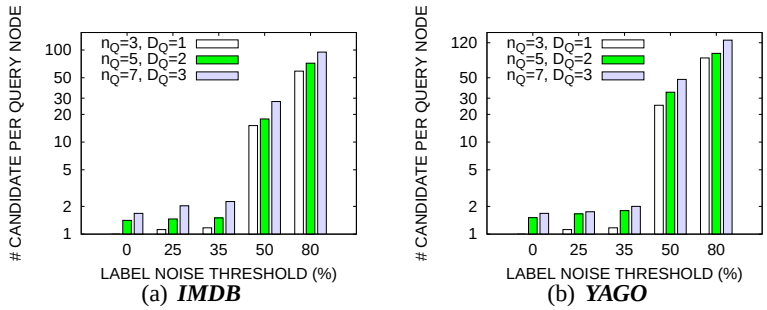
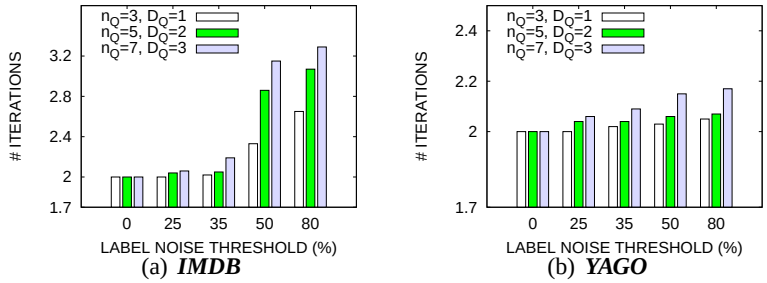


Figure 2.8: NeMa: Query Performance vs. Noise; YAGO; n_Q: Query Nodes, D_Q: Query Diameter



(a) ***IMDB*** (b) ***YAGO***

Figure 2.9: NeMa: # Candidates vs. Label Noise



(a) ***IMDB*** (b) ***YAGO***

Figure 2.10: NeMa: # Iterations vs. Label Noise

Comparing Methods. We compared NeMa with keyword search (BLINKS [76]) and various graph querying methods: SAGA [142], IsoRank [138], NESS [95], and NeMa_{gs} - a variation of NeMa following gStore [171]. All these methods were implemented in C++.

In our experiments, (1) propagation factor α and depth of vectorization h (Section 2.3) were set as 0.5 and 2, respectively [95], (2) the optimal values of the proportionality constant λ (Eq. 2.4) for different datasets were obtained empirically (Figure 2.5(a)), (3) the indexes were stored in the hard disk. All the experiments were run using a single core in a 100GB, 2.5GHz Xeon server.

2.7.2 Effectiveness and Efficiency

Performance over Real-life Data Sets

In these experiments, we evaluated the performance of of NeMa over three real-life graphs, averaged over 100 queries (Figure 2.5). For each target graph, we randomly generated 100 query graphs with $|V_Q|$ = 7 and D_Q = 3. We fixed the structural noise as 30%, label noise as 50%, and label noise threshold as 50%.

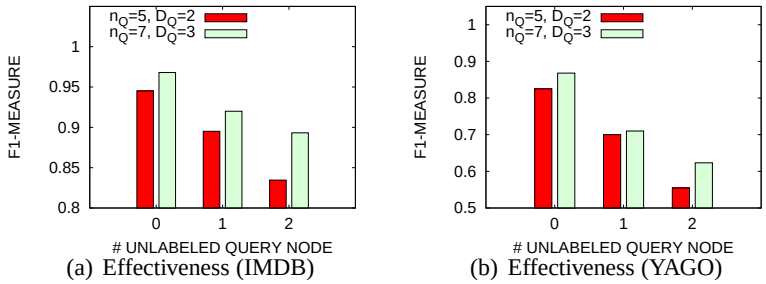


Figure 2.11: NeMa: Effectiveness with Unlabeled Query Nodes

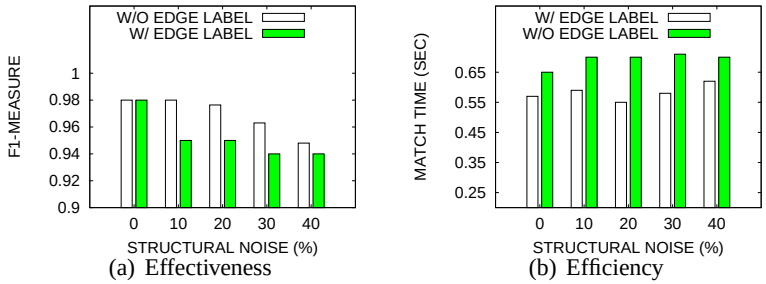


Figure 2.12: NeMa: Performance with Edge Labels (IMDB)

Figure 2.5(a) shows the effectiveness of NeMa over various datasets, and with different values of the proportionality constant λ. For all the three datasets, our algorithm *always* correctly identifies more than 76% of the query nodes. Specifically, the F1-measure is 0.94 for *IMDB*, with $\lambda = 0.3$, even when we introduced 30% structural noise and 50% label noise.

The effectiveness is higher over *IMDB* due to the type constraint posed with the query nodes. Besides, the optimal value of λ lies between $0.3 \sim 0.5$ in these datasets.

Figure 2.5(b) reports the efficiency of NeMa using the same setting as in Figure 2.5(a), including the running time of off-line index construction (INDEX) and online query evaluation (MATCH). We observed the following. (a) NeMa identifies the best match in less than 0.2 seconds, over all three datasets [6]. (b) The top-k match finding time does not vary significantly, over different k, since our inferencing method is always executed once. (c) The time required for indexing is modest (e.g., 9862 sec for the *YAGO* dataset with 13M nodes and 18M edges). (d) The indexing and querying times are longer over *IMDB*, due to its higher density.

Performance against Noise

In this set of experiments, we investigated the impact of varying noises on the performance of NeMa. Three sets of query graphs were generated by setting (i) $|V_Q| = 3$, $D_Q = 1$, (ii) $|V_Q| = 5$, $D_Q = 2$, and (iii) $|V_Q| = 7$, $D_Q = 3$. Under each query set, 100 query graphs were generated.

Varying label noise. Fixing the structural noise as 30%, we varied the label noise from 0% to 50%, and investigated the effectiveness of NeMa, when the label noise threshold was set as 35% and 50%, respectively. As shown in Figure 2.6(a) over *DBpedia*, (1) the F1-measure decreases as the label noise increases, since the candidate set of each query node may contain more candidates that are not true matches, which in turn reduces the effectiveness, (2) the F1-

[6] our indexing and matching algorithm can be parallelized for every node. Hence, one may implement NeMa in a PREGEL [115] platform, for larger graphs.

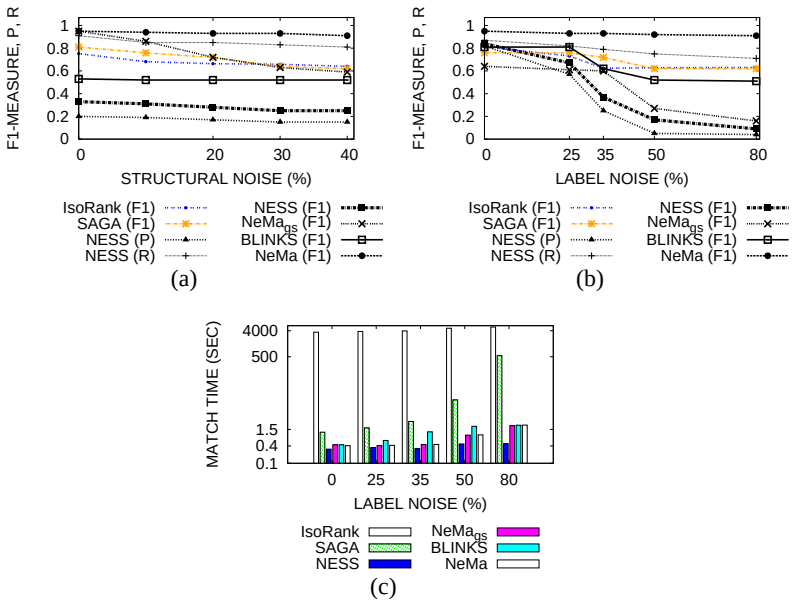


Figure 2.13: Comparison Results (IMDB): NESS, BLINKS Modified for Approximate Label Match. NESS Results Correspond to its *Filtering* Phase.

measure is higher when the label noise threshold is higher, since the candidate sets are more likely to include the correct matches. Observe that the F1-measure is always above 0.60.

Figure 2.6(b) shows that NeMa efficiency is insensitive to label noise, but more sensitive to label noise threshold. This is because it takes NeMa more time to process the larger candidate sets for the query graph as the label noise threshold increases.

Varying structural noise. Fixing the label noise threshold as 35% and label noise as 35%, we varied the structural noise from 0% to 40% in Figure 2.7(a) and 2.7(b). It can be observed that (a) both effectiveness and efficiency decrease as we increase the structural noise, and (b) with the increase of the query size, both effectiveness and efficiency increase. The reason is that (1) larger queries have more constraints in the neighborhood of a query node, which benefit the identification of correct matches, and (2) it takes longer time for NeMa to compare the matching

cost for larger queries. Moreover, the F1-measure is always above 0.93, and the running time is always less than 0.1 seconds, even with 40% structural noise.

Varying label noise threshold. Fixing the structural noise as 30% and the label noise as 35%, we investigated the effect of varying the label noise threshold on the query performance.

The effectiveness and efficiency of NeMa over *IMDB* are illustrated in Figures 2.7(c) and 2.7(d), respectively. We observed the following. (1) The F1-measure initially increases while we increase the label noise threshold. This is because the query node labels are updated by adding random words. Hence, the higher is the label noise threshold, there is more chance that the correct match of a query node will be selected in its candidate set. (2) When the label noise threshold is more than 35%, the F1-measure does not improve significantly. Therefore, the optimal value of the label noise threshold can be determined empirically based on the query and target graphs. On the other hand, the running time of NeMa increases with the increase of the label noise threshold. This is because (a) the candidate matches per query node increases (see Figure 2.9), and (b) the number of iterations required for the convergence of our network matching algorithm also increases (see Figure 2.10). Hence, it takes more time for NeMa to find the matches.

Effectiveness with Unlabeled Query Node

We next verified the effectiveness of NeMa in the presence of *unlabeled nodes* in query graphs. These experiments simulate RDF query answering (see Section 2.5.2). For these experiments, we randomly selected two sets of 100 query graphs each, where (i) $|V_Q| = 7$,

$D_Q = 3$, and (ii) $|V_Q| = 5$, $D_Q = 2$, respectively. Fixing the structural noise as 30%, label noise as 35%, and label noise threshold as 35%, we varied the number of unlabeled query nodes from 0 to 2. As shown in Figure 2.11, (1) the F1-measure decreases over both datasets while the number of the unlabeled nodes increases, because the unlabeled nodes introduce more candidates, which in turn reduces the effectiveness, (2) the effectiveness is higher over *IMDB* due to the type constraints, and (3) over all the cases, the F1-measure is always above 0.50.

Performance with Propagation Depth

In these experiments, we analyzed the effect of propagation depth h in our query performance. We randomly selected 100 query graphs from *YAGO*, with $|V_Q| = 7$, $D_Q = 3$, structural noise 30%, label noise 50% and label noise threshold 50%. Table 2.4 shows that the efficiency of NeMa decreases with increasing h, especially the index time increases exponentially with h. However, for $h = 2$, we obtained an acceptable F1-measure of 0.86.

	$h = 1$	$h = 2$	$h = 3$
Index Time (sec)	265	9862	236553
Match Time (sec)	0.58	0.92	2.76
F1-Measure	0.61	0.86	0.87

Table 2.4: NeMa: Query Performance with Varying h (YAGO)

Performance with Edge Labels

We verified the query performance in the presence of edge labels (Section 2.5.2). We randomly selected 100 query graphs from *IMDB*, with with $|V_Q| = 7$, $D_Q = 3$, label noise 50% and label noise threshold 50%. We varied the structural noise from 0% to 40%. The labels

of the newly inserted edges in the query graphs were assigned by generating random strings. Figure 2.12 shows that, (1) when no structural noise is added, the F1-measure remains same for both the cases of labeled and unlabeled edges. (2) However, with the addition of structural noise, the F1-measure decreases slightly for the case of labeled edges, since there are more noises in the query graphs due to the labels of newly inserted edges. (3) The running time is higher for the case of labeled edges because additional time is required to measure edge label similarities.

Comparison with Existing Algorithms

We compared the performance of NeMa with IsoRank [138], SAGA [142], NESS [95], gStore [171], and BLINKS [76]. (1) IsoRank and SAGA find optimal graph matches in smaller biological networks considering structure and node label similarities. (2) NESS finds the top-k graph matches from a large network, but with strict node label equality. Hence, we modified NESS by allowing two nodes to be matched if their label difference is within the label noise threshold. (3) We considered a variation of NeMa, namely, NeMa_{gs}, which allows label difference but resorts to strict isomorphic matching. Thus, NeMa_{gs} essentially follows the same principle as gStore, which is a subgraph isomorphism-based SPARQL query evaluation method with node label differences. (4) BLINKS [76], a keyword search method, supports only structural mismatches. Hence, we also modified BLINKS by allowing node label differences within the label noise threshold.

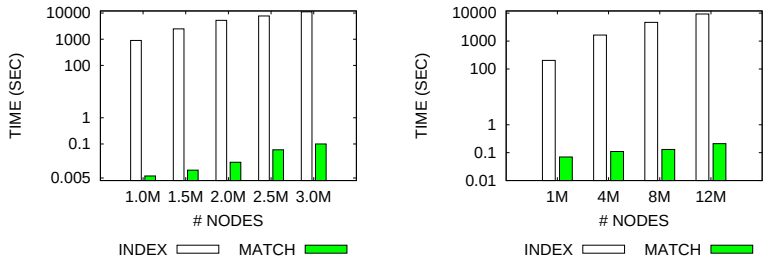
All these methods, except NESS, find the top-1 graph match directly. Hence, we considered the top-1 match corresponding to each query node to evaluate precision, recall, and F1-measure; and thereby obtained the same score for them. In contrast, NESS employs a *filtering-and-verification* approach, where its filtering phase reports a set of high-quality final candidate nodes for each query node. Then, it verifies all possible graph matches formed by these final candidate nodes, in order to find the top-1 graph match. Therefore, we report precision, recall and F1-measure of its filtering phase, which is the most important step in NESS. For fairness, we reported only the running time of its filtering phase in Figure 2.13(c).

For these experiments, we randomly selected 100 query graphs, where $|V_Q|$ = 7 and D_Q = 3, using the *IMDB* dataset. In each query graph, one node was *unlabeled* and the labels of the remaining nodes were updated by randomly inserting new words. We varied structural noise in Figure 2.13(a), and fixed both label noise and label noise threshold as 50%. Observe that, with no structural noise, NeMa and NeMa$_{gs}$ have F1-measure about 0.94; but with the increase in structural noise, NeMa (F1-measure 0.9) outperforms the other methods (F1-measure $0.1 \sim 0.7$)

We varied label noise and label noise threshold in Figures 2.13(b) and 2.13(c), and fixed structural noise as 30%. The label noise threshold had the same value as label noise in these figures. With no label noise, NeMa has F1-measure 0.93, whereas NESS, IsoRank, SAGA, and BLINKS have F1-measures about 0.8. However, as we increase label noise, NeMa (F1-measure 0.9) outperforms the other methods (F1-measure $0.1 \sim 0.7$) by a large margin.

NeMa finds the best match in less than 1 sec, while IsoRank takes 5000 sec. SAGA requires 15 sec and 546 sec, with label noise 50% and 80%, respectively.

2.7.3 Scalability



(a) ***IMDB*** (b) ***YAGO***

Figure 2.14: NeMa: Scalability

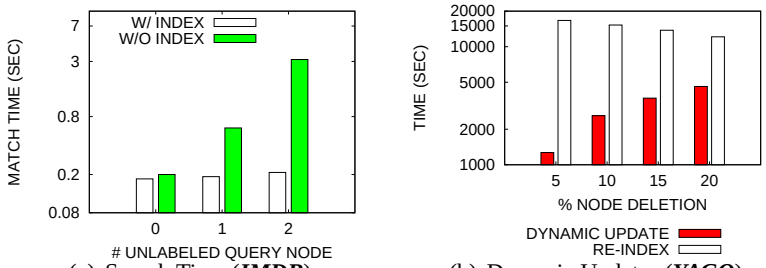
In this section, we analyzed the scalability of NeMa by varying the number of nodes in the *YAGO* and *IMDB* networks. We used 100 randomly selected query graphs, where $|V_Q|$ = 7, d_Q = 3, and fixed the structural noise as 30%, label noise as 35%, and label noise threshold as 35%. Figure 2.14 shows that NeMa scales well with the size of the target graphs. Specifically, the off-line indexing time increases polynomially, and the online query evaluation time linearly with the increase of the size of the target networks.

2.7.4 Optimization Techniques

In these experiments, we investigated the performance of the optimization techniques of NeMa. We randomly selected 100 query graphs, where $|V_Q|$ = 7 and D_Q = 3, and fixed the structural noise as 30% and both label noise and label noise threshold as 35%. In each query graph, the number of *unlabeled* query nodes is varied from 0 to 2. Figure 2.15(a) shows that the

indexing and optimization techniques significantly improve the efficiency of NeMa, specifically by a factor of 15 in the presence of 2 unlabeled query nodes.

We also compared the index construction time with *dynamic update* against the cost of rebuilding the whole index. In these experiments, we considered only deletion of nodes (and thereby, deletion of the incident edges) from the original network as a method of dynamic updates. Figure 2.15(b) shows that, for a wide range of updates in the target graph, it is more efficient to update the index structure rather than re-indexing the graph.



(a) Search Time (***IMDB***) (b) Dynamic Updates (***YAGO***)

Figure 2.15: NeMa: Index Performance

2.8 Summary

In this chapter, we have introduced NeMa, a novel graph querying framework via subgraph matching that allows for ambiguity in both structure and node labels. We convert the neighborhood of each node into a multi-dimensional vector, and then apply an inference algorithm to identify the optimal graph matches. We further investigate how NeMa can be extended to various graph query-processing applications, such as RDF query answering, graph matching with edge labels, and finding top-k approximate matches. Our experimental results over real-life datasets show that NeMa efficiently finds high-quality matches, as compared to

state-of-the-art graph querying methods. In future work, one may consider approximate sub-graph matching over graph streams, and also more sophisticated label similarity metrics, e.g., ontology and semantic similarity.

Chapter 3

Reliability in Uncertain Graphs

> *" ... as far as the propositions of mathematics refer to reality, they are not certain; and as far as they are certain, they do not refer to reality."*
>
> Albert Einstein

Due to noisy measurements, inference errors, and other causes, in many emerging application domains data are represented in the form of uncertain graphs, that is graphs whose arcs are associated with a probability of existence. A fundamental problem on such graphs is to compute the *reliable set* RS(S,η)—the set of all nodes that are reachable from a query set of nodes S with probability no less than a given threshold η. Reliable set computation is a generalization of the source-to-target reliability problem, which is known to be **#P**-complete. Traditional techniques for computing source-to-target reliability apply sampling methods, which are not efficient enough for the generalized version of the problem.

In this chapter, we propose RQ-tree, a novel index for efficiently estimating the reliable set, which is based on a hierarchical clustering of the nodes in the graph, and fur-

ther optimized using balanced minimum cut techniques. Based on RQ-tree, we define a fast filtering-and-verification online query evaluation strategy that relies on a maximum-flow-based candidate-generation phase, followed by a verification phase consisting of either a lower-bounding method or a sampling technique. The first verification method does not return any incorrect node, thus guaranteeing perfect precision, completely avoids sampling, and is more efficient. The second verification method ensures instead better recall.

Extensive experiments on real-world uncertain graphs and under several settings show that our approach is very efficient—speed-up over sampling methods up to six orders of magnitude, as well as accurate—recall typically in the $[0.75, 0.98]$ range.

3.1 Introduction

Recent advances in social and information science have shown that linked data is pervasive in our society and the natural world around us. Therefore, graphs have become ubiquitous models to represent such complex structured data. However, in many novel applications, uncertainty is inherent in the data due to a variety of reasons, such as noisy measurements [9], inference and prediction models [5, 103], or explicit manipulation, e.g., for privacy purposes [26]. In these cases, data can be represented as an *uncertain graph*, also called probabilistic graph, i.e., a graph whose arcs are labeled with a probability of existence.

A fundamental primitive on uncertain graphs is given by *reliability queries*, whose main goal, generally speaking, is computing the probability that any two (sets of) nodes are reachable. Most attention has been so far devoted to the simplest version of such queries,

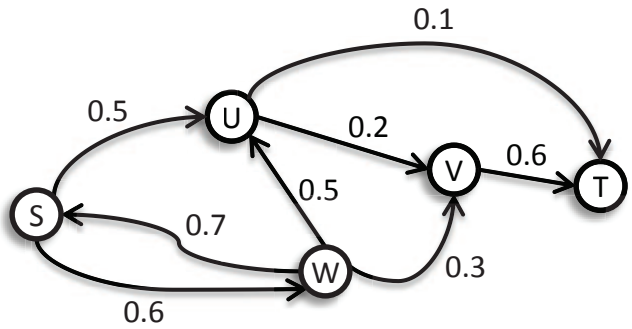


Figure 3.1: Run-Through Example: An Uncertain Graph.

that is *source-to-target reliability*, where one is asked to compute the probability that a *single* target node is reachable from a *single* source node. Source-to-target reliability is a prototypical **#P**-complete problem [20, 148], thus the focus in this regard has been on developing effective approximation methods. Most of the work has concerned Monte-Carlo sampling methods [61, 92, 131], which have a major efficiency shortcoming due to the large number of samples that is typically needed to obtain satisfactory accuracy. To alleviate this issue, some smarter sampling methods have been proposed recently [87, 170].

In this chapter, we study a more general reliability problem, which we call *reliable set computation*. Instead of focusing on the reliability between any two single nodes, we shift the attention on *sets of nodes*: given a probability threshold $\eta \in (0, 1)$ and a *set* of source nodes S, compute *all* nodes that are reachable from S with probability no less than η. The generalization with respect to source-to-target reliability is twofold: we allow *multiple source* nodes, while also asking for *all* target nodes that satisfy the query. To the best of our knowledge, this is the first work focusing on such a generalized reliability problem.

Example 3.1 (Reliable Set Computation)**.** *Consider the uncertain graph in Figure 3.1, and suppose one wants to compute all nodes that are reachable from* $\{\mathtt{s}\}$ *with probability greater*

than 0.5. *We denote this query as* RS({s},0.5). *It is easy to verify that* w *is part of the solution due to a direct arc from* s *with probability greater than the query threshold. Additionally,* u *can be reached from* s *directly, or via* w. *Thus, the probability that* u *is reachable from* $\{s\}$ *is equal to the probability that at least one among the direct path and the path through* w *exists. Assuming independence among the existence of the arcs in the graph, this probability is* $1 - (1 - 0.5) \times (1 - 0.6 \times 0.5) = 0.65$. *Hence,* u *also belongs to the solution set.*

Node v *is reachable if at least one of the following events holds: (1) the arcs* (s,u), (u,v) *exist; (2) the arcs* (s,w), (w,v) *exist; (3) the arcs* (s,w), (w,u), (u,v) *exist. However, the three events are not independent. Hence, to properly do the computation, one has to consider all possible instances of this subgraph consisting of* 5 *arcs, which are* $2^5 = 32$. *Note that even for such a small graph, it is non-trivial to determine the probability that* v *is reachable from* s. *For* t, *the computation becomes even more convoluted. Considering all possible instances of the uncertain graph, one may verify that the answer to our query is:* RS({s},0.5) = {s,u,w}.

Applications. The reliable set computation problem naturally arises in a variety of scenarios. In the problem known as *influence maximization*, whose main application is *viral marketing* [92], the probability of an arc (u, v) represents the influence that u exerts on v, i.e., the likelihood that some action of u will be adopted by v, or the likelihood that information propagates from u to v. An important, as well as the most computationally expensive step common to the state-of-the-art techniques is to determine *all* nodes that can be influenced by a given *set* of seed nodes: in Section 3.7.6, we show how this can be approximated efficiently by means of reliable-set-computation queries.

In protein-protein interaction networks [100], nodes represent proteins and arcs represent *interactions* among them. Interactions are established for a limited number proteins, through noisy and error-prone experiments. Therefore, each arc is associated with a probability accounting for supporting evidence on the existence of the interaction. In this context, predicting co-complex memberships [17, 100], and new interactions [131, 136] require to compute all proteins that are evidently (i.e., with high probability) reachable from a core (source) set of proteins.

In mobile ad-hoc networks, the connectivity between nodes is estimated using noisy measurements, and the notion of "delivery probability" can be used to quantify the probability that a given node can deliver a packet to another node [68]. Similarly, road networks can be modeled as uncertain graphs due to unexpected traffic jams [80]. In such types of networks, reachability from a set of alternative source locations to a set of affordable target locations should be interpreted in a probabilistic way, by means, i.e., of reliable-set-computation queries.

In all applications such as those listed above, the rate of reliable set computations to be performed is usually very high. Thus, a fundamental requirement is to perform any single reliable set computation very quickly. This makes Monte-Carlo sampling not suitable: such methods need in fact to (entirely) visit multiple graph instances sampled from the input uncertain graph, and the number of such samples is typically large in order to guarantee reasonable accuracy. Moreover, the generalizations of our problem make the smarter sampling methods by Jin et al. [87] and by Zhu et al. [170] not well-suited either. Indeed, one needs to apply those methods once for every other node in the input graph to find the whole reliable set, which is

Table 3.1: Time Complexity of Reliable Set Computation

RQ-tree vs. existing source-to-target reliability methods when applied as baselines for reliable set computation. n and m are the number of nodes and arcs of the input uncertain graph $\mathcal{G}$, d is the diameter of $\mathcal{G}$, K is the number of deterministic graphs sampled from $\mathcal{G}$, S is the set of query source nodes. $\tilde{n}$ and $\tilde{m}$ ($\tilde{n} \ll n$, $\tilde{m} \ll m$) are the number of nodes and arcs of the subgraph of $\mathcal{G}$ that RQ-tree needs to visit.

	MC-Sampling [61]	*RHT-Sampling* [87]	RQ-tree-LB (this work)	RQ-tree-MC (this work)
single-source queries	$\mathcal{O}(K(m+n))$	$\mathcal{O}(n^2 d)$	$\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\tilde{n}\tilde{m} + K(\tilde{m}+\tilde{n}))$
multiple-source queries	$\mathcal{O}(K(m+n))$	$\mathcal{O}(n^2 d)$	$\tilde{\mathcal{O}}(\lvert S\rvert\tilde{n}\tilde{m})$	$\tilde{\mathcal{O}}(\lvert S\rvert\tilde{n}\tilde{m} + K(\tilde{m}+\tilde{n}))$

very inefficient in large graphs (Table 3.1). It is apparent, therefore, that the generalizations we introduce in this work are non-trivial and make the reliability problem much harder.

Within this view, we address the problem of fast online estimation of reliable set queries by pre-computing offline information that can be profitably exploited to speed-up online query processing. We introduce a novel index, called RQ-tree, which allows to process our queries very efficiently—up to six orders of magnitude faster than sampling methods. Our offline indexing technique is based on a *hierarchical clustering* of the nodes in the input graph, and further optimized with balanced-minimum-cut techniques. Query evaluation consists of a maximum-flow-based candidate generation (filtering) step and a verification step that relies on either (a) an efficient lower bound based on the notion of most-likely path, or (b) a sampling technique applied to the candidate set only. The former verification method guarantees perfect

precision, as it returns no incorrect (false positive) nodes (while false negatives can arise); it also avoids sampling at all, resulting in very high efficiency. In the sampling-based verification method, our RQ-tree forms the basis to speed-up any sampling method for reliable set computation, as it allows for significantly reducing the size of the subgraph to be sampled. Sampling-based verification guarantees in general better accuracy in terms of recall.

Summary of contributions and roadmap. We achieve the following contributions:

- We define the fundamental problem of computing the reliable set in uncertain graphs (Section 3.3).
- For retrieving the reliable set, we develop an efficient index, called RQ-tree, which is based on a hierarchical clustering of the nodes in the graph (Section 3.4).
- Based on RQ-tree, we develop a fast filtering-and-verification strategy. We derive an upper bound for the probability of a set of nodes contained in a cluster to reach nodes outside the cluster (Section 3.5.1), and a lower bound for the probability of reaching any other node (Section 3.5.2). The first bound is used in the candidate generation (filtering) phase, while the second bound is used for verification.
- We propose a balanced-minimum-cut-based partitioning method to build the RQ-tree index (Section 3.6).
- We conduct a thorough experimental evaluation by involving several real-world uncertain graphs and comparing RQ-tree with a couple of baselines: a Monte-Carlo-sampling technique [61] and the sampling method by Jin et al. [87] designed for source-to-target re-

liability (Section 3.7). Results attest the high efficiency and accuracy of RQ-tree, as well as its superiority with respect to the baselines.

- We show application of RQ-tree in the influence-maximization problem [92], which requires to find a set of influential users that maximize the spread of an information in social networks (Section 3.7.6).

3.2 Related Work

Probabilistic Reachability Queries. In many application domains, such as social, biological, and mobile networks, uncertain graphs have received a great deal of attention in the last years. Traditional approaches to probabilistic reachability queries over uncertain graphs rely on Monte-Carlo sampling methods [61]. Zhu et al. [170] focus on threshold-based probabilistic reachability and derive an upper bound for reachability that can be used as a pruning rule. When such a rule cannot be applied, a dynamic Monte-Carlo sampling technique is employed. Jin et al. [87] deal with distance-constrained reachability queries: given two nodes s and t, what is the probability that the distance from s to t is less than or equal to a user-defined threshold? We remark that both these works [87, 170] consider reachability from a *single source* node to a *single target* node, while we generalize the problem allowing multiple source and target. As shown in our experiments, source-to-target approaches are not well-suited for reliability queries over large graphs, as they need to be executed from scratch for every node as a target node in the input graph to build the whole reliable set desired.

Influence Maximization. Influence maximization aims at finding a set of seed nodes that generate the largest expected information cascade: the uncertainty in the graph represents the influence. Domingos and Richardson [52] have formulated influence maximization as an optimization problem, while [92] defines approximation algorithms with provable performance guarantees. Several heuristics have been also proposed to improve the efficiency of that method [37, 38, 71, 108]: some of these can easily be coupled with an RQ-tree index to further improve their speed-up.

Indexing for Reachability Queries. Reachability queries between a source and a target node in deterministic graphs have been widely studied. Several indexing methods are also proposed, e.g., interval labeling [10, 145, 163], transitive closure [14, 88], 2-hop indexing [42, 134], and other compression based methods [56, 57, 149]. To the best of our knowledge, ours is the first work that proposes an indexing method for the generalized reliability queries over large uncertain graphs.

3.3 Problem Statement

An *uncertain graph* $\mathcal{G}$ is a triple (N, A, p), where N is a set of n nodes, $A \subseteq N \times N$ is a set of m directed arcs, and $p : A \rightarrow (0, 1]$ is a probability function that assigns a probability of existence to each arc in A.

Almost all works on uncertain graphs assume the probabilities of existence of the arcs in the graph independent from one another [87, 131, 172]. Particularly, the well-known *possible-world semantics* is usually adopted: an uncertain graph $\mathcal{G}$ with m arcs yields 2^m pos-

sible deterministic graphs, which are derived by sampling independently each arc $a \in A$ with probability $p(a)$. More precisely, a possible graph $G \sqsubseteq \mathcal{G}$ is a pair (N, A_G), where $A_G \subseteq A$, and its sampling probability is:

$$\Pr(G) = \prod_{a \in A_G} p(a) \prod_{a \in A \setminus A_G} (1 - p(a)). \tag{3.1}$$

For any possible deterministic graph (world) G, we define an indicator function $P_G(S, t)$ to be 1 if there exists a path in G from a set of source nodes $S \subseteq A$ to a target node $t \in A$, and 0 otherwise. Here, we say that a path from the set of nodes S to a target node t exists if there exists a path from *at least* one node $s \in S$ to t. The probability that t is reachable from S in the uncertain graph $\mathcal{G}$, denoted by $R(S, t)$, can be computed as:

$$R(S, t) = \sum_{G \sqsubseteq \mathcal{G}} P_G(S, t) \Pr(G). \tag{3.2}$$

The number of possible worlds $G \sqsubseteq \mathcal{G}$ is exponential in the number of arcs, which makes the exact computation of $R(S, t)$ infeasible even for moderately-sized graphs.

The problem we address in this work is the following.

Problem 3.1 (Reliable Set Computation)**.** *Given an uncertain graph $\mathcal{G} = (N, A, p)$, a probability threshold $\eta \in (0, 1)$, and a set of source nodes $S \subseteq N$, find all nodes in N that are reachable from S with probability greater than or equal to η, that is, $RS(S, \eta) = \{t \mid t \in N, R(S, t) \geq \eta\}$.*

Our problem generalizes the source-to-target reliability problem, which asks to compute the probability that a path from a source node s to a target node t exists. The source-to-target reliability problem is a prototypical **#P**-complete problem [20, 148]. It is easy to verify

that one can obtain a reduction from the source-to-target reliability problem to Problem 3.1. The idea of the reduction is to perform a binary search on the threshold probability η in order to estimate the answer to a given instance of the source-to-target reliability problem. As a result, Problem 3.1 is hard as well.

3.4 The RQ-Tree index: Overview

The reliable set computation problem generalizes source-to-target reliability by shifting the focus from single nodes to sets of nodes. For this purpose, it is natural for any index structure dealing with such a generalized version of the problem to pre-compute and store information in terms of sets of nodes. This way, during query evaluation, one can retrieve and further process the appropriate node set only, instead of the whole graph. To achieve this goal, our intuition is to define the proposed RQ-tree index based on a *hierarchical clustering* of the nodes in the input uncertain graph. The RQ-tree, hereinafter denoted by $\mathcal{T}$, is a tree, where the root contains the complete set of nodes N, and the leaves correspond to individual nodes of N. All clusters at any level i form a partition of N. A cluster at level i is partitioned into a number of children clusters at level $i+1$. As a result, there exists a unique path in $\mathcal{T}$ that connects each node $s \in N$ to the root. Such a path is composed of clusters that are all nested into each other. An example of RQ-tree index for the uncertain graph of Figure 3.1 is shown in Figure 3.2, together with some bounds that will be clarified in the next section.

Our query-processing strategy is based on two phases:

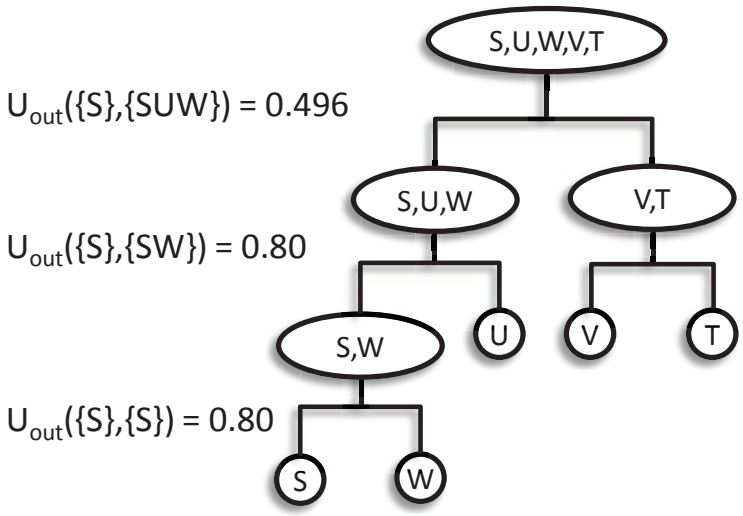


Figure 3.2: An RQ-tree Index for the Uncertain Graph in Figure 3.1

1. *Candidate generation*, where a *candidate* set of nodes is built based on the information stored into the pre-computed RQ-tree index. All nodes not belonging to the candidate set are discarded. A nice feature of this step is to guarantee that *no true positive* node is discarded from the candidate set.

2. *Verification*, where a screening is applied to the candidate set in order to filter out nodes that should not be part of the answer.

The specific way of defining the RQ-tree hierarchical clustering mainly depends on the proposed RQ-tree-based query processing, as a major desideratum of this work is to achieve high query processing performance in terms of both efficiency and accuracy. For the sake of clarity, hence, in the following we first discuss the details of our query-processing strategy assuming an RQ-tree given (Section 3.5). Then, in Section 3.6, we describe how to build the RQ-tree index in order to obtain fast and effective query processing.

3.5 Query Processing

3.5.1 Candidate Generation

Here we describe the candidate-generation step of our online reliable-set computation strategy. We start by presenting the main theoretical results that form the basis of the proposed method (Section 3.5.1). Then, we describe the method in details: for the sake of clarity, we first discuss the case in which the source set is a singleton (Section 3.5.1), then we discuss the general case where the source set has cardinality larger than one (Section 3.5.1).

Upper bound on outreach probability

A key concept in our candidate-generation algorithm is the notion of *outreach probability*, which is the probability that a subset of nodes S within a cluster C in the RQ-tree index is connected to nodes outside C, i.e., within $\overline{C} = N \setminus C$.

Definition 3.1 (outreach probability)**.** *Given a set of nodes (cluster) $C \subseteq N$ and a subset $S \subseteq C$, the* outreach probability $R_{out}(S, C)$ *from S to outside C is defined as the probability that S reaches the nodes not belonging to C, i.e.,*

$$R_{out}(S, C) = \sum_{G \sqsubseteq \mathcal{G}} P_G(S, \overline{C}) \Pr(G) \tag{3.3}$$

where $P_G(S, \overline{C}) = 1$ if there exists at least a node $t \in \overline{C}$ such that $P_G(S, t) = 1$, $P_G(S, \overline{C}) = 0$ otherwise.

A straightforward feature of outreach probability is the following: if the outreach probability of S in C is smaller than a certain value η, then the probability of reaching *every* node t outside C is also smaller than η.

Theorem 3.1. *For a cluster $C \subseteq N$ and its subset $S \subseteq C$ the following holds: $R_{out}(S, C) < \eta \Rightarrow R(S, t) < \eta$, for all $t \in \overline{C}$.*

Proof By definition, $R_{out}(S, C)$ is the probability that at least one node $s \in S$ can reach *at least one* node $t \in \overline{C}$. Clearly, this probability is greater than or equal to the probability that at least one node $s \in S$ can reach *any specific single* node $t \in \overline{C}$. In other words, $R_{out}(S, C) \geq R(S, t)$, for all $t \in \overline{C}$, which implies the theorem. □

Our query processing relies on Theorem 3.1, as well as on the next theorem, which relates the outreach probabilities of any two clusters that are nested into each other.

Theorem 3.2. *Given any two clusters C_i, C_j such that $C_i \subseteq C_j$, and a set of source nodes $S \subseteq C_i$, it holds that $R_{out}(S, C_i) \geq R_{out}(S, C_j)$.*

Proof If $C_i \subseteq C_j$, then $\overline{C_i} \supseteq \overline{C_j}$. Therefore, in all possible worlds $G \sqsubseteq \mathcal{G}$ in which a path from S to $\overline{C_j}$ exists, there also exists a path from S to $\overline{C_i}$. The theorem follows trivially from the definition of $R_{out}(S, C)$. □

Theorems 3.1 and 3.2 create the basis for retrieving a valid candidate set from an RQ-tree $\mathcal{T}$. Specifically, given a query $RS(S, \eta)$, consider all clusters C in $\mathcal{T}$, such that, $S \subseteq C$ and $R_{out}(S, C) < \eta$. Theorem 3.1 guarantees that all nodes outside each of those clusters violate the reliability condition, therefore they can be safely discarded. Clearly, one wants to

consider only the smallest among those clusters in order to maximize the number of pruned nodes. Theorem 3.2 ensures that one only needs to focus on the cluster C having the largest value $R_{out}(S, C)$ that is smaller than η.

A candidate-generation strategy based on the above reasoning would require to compute outreach probabilities exactly, which is infeasible as it would need to enumerate all possible worlds. A possible solution is to approximate R_{out} values by means of sampling. Unfortunately, besides the well-known efficiency issues, this sampling-based solution would not guarantee that the results stated in Theorems 3.1–3.2 carry over. For this purpose, we derive an upper bound for R_{out}, which can be efficiently computed without sampling for any subset of nodes $S \subseteq C$. The proposed upper bound, denoted by U_{out}, is based on the min-cut/max-flow principle. We start by defining the notion of *most-likely cut* between two disjoint sets of nodes.

Definition 3.2 (most-likely cut)**.** *Consider a deterministic graph $G = (N, A)$ and two disjoint sets of nodes $X, Y \subseteq N$. We define a cut $\mathcal{C}(X, Y)$ between the sets X and Y to be a set of arcs in A whose removal disconnects X and Y. Consider now an uncertain graph $\mathcal{G} = (N, A, p)$ and two disjoint sets of nodes $X, Y \subseteq N$. We define the most-likely cut $\mathcal{C}^*(X, Y)$ to be a set of arcs such that: (1) it is a cut between X and Y, as defined on the deterministic graph that contains all the arcs of $\mathcal{G}$; (2) among all cuts between X and Y, it is the one that maximizes the probability of having all its arcs non-present, i.e., $\mathcal{C}^*(X, Y) = \arg\max_{\mathcal{C}(X,Y)} \prod_{a \in \mathcal{C}(X,Y)} (1 - p(a))$.*

As stated in the following theorem, the most-likely cut provides us a way to express the desired upper bound U_{out}.

Theorem 3.3. *Given a cluster $C \subseteq N$ and a subset $S \subseteq C$, it holds that:*

$$R_{out}(S,C) \leq U_{out}(S,C) = 1 - \max_{\mathcal{C}(S,\overline{C})} \prod_{a \in \mathcal{C}(S,\overline{C})} (1 - p(a)).$$

Proof Consider any cut $\mathcal{C}(S,\overline{C})$. From the independence assumption, the probability that none of the arcs in $\mathcal{C}(S,\overline{C})$ exists is equal to $\prod_{a \in \mathcal{C}(S,\overline{C})}(1-p(a))$. Now, consider the event that none of the nodes in S can reach any node outside C. The probability of such an event is equal to $1 - R_{out}(S,C)$, and is clearly lower-bounded by the probability that no arc in $\mathcal{C}(S,\overline{C})$ exists. Based on this reasoning, it holds that:

$$1 - R_{out}(S,C) \geq \prod_{a \in \mathcal{C}(S,\overline{C})} (1 - p(a)), \text{ for all } \mathcal{C}(S,\overline{C}),$$

or equivalently,

$$R_{out}(S,C) \leq 1 - \max_{\mathcal{C}(S,\overline{C})} \prod_{a \in \mathcal{C}(S,\overline{C})} (1 - p(a)).$$

The theorem follows. □

The upper bound U_{out} defined in Theorem 3.3 can be computed by executing a max-flow algorithm on a capacitated graph appropriately derived from $\mathcal{G}$. Specifically, our algorithm works as follows (see pseudocode in Algorithm 1). First, we construct a capacitated graph $\widehat{G}$, which has the same sets of nodes and arcs as $\mathcal{G}$. Each arc a in $\widehat{G}$ has a capacity $c(a)$ equal to $-\log(1 - p(a))$. Then, we compute the max-flow f^* from S to $\overline{C}$ on $\widehat{G}$. Note that the algorithm exploits the following observation: the max flow from S to $\overline{C}$ is equivalent to the max flow from S to the set $\overline{C}' \subseteq \overline{C}$ of all nodes in $\overline{C}$ having an incident arc outgoing from S, i.e., the set $\overline{C}' = \{v \in \overline{C} \mid \exists\, u \in S \ : \ (u,v) \in A\}$. Thanks to this observation, the max-flow computation can be significantly speeded-up, as typically $|\overline{C}'| \ll |\overline{C}|$.

Algorithm 1 Compute outreach probability bound

Input: an uncertain graph $\mathcal{G} = (N, A, p)$; a cluster $C \subseteq N$; a set of source nodes $S \subseteq C$

Output: $U_{out}(S, C)$

1: $\overline{C}' \leftarrow \{v \in \overline{C} \mid \exists\, u \in S \;:\; (u, v) \in A\}$

2: for all $a \in A$, set $c(a) = -\log(1 - p(a))$

3: build $\widehat{G} = (N, A, c)$

4: $f^* \leftarrow$ MaxFlow($\widehat{G}, S, \overline{C}'$)

5: $U_{out}(S, C) \leftarrow 1 - \exp(-f^*)$

Moreover, recall that the max-flow problem is defined between a single source and a single sink node. In our case, we ask for the max-flow between a set of source nodes S and a set of sink nodes $\overline{C}'$. This generalized variant can be easily mapped to the base case of single-source single-sink: the idea is to create a dummy source s_0 and a dummy sink t_0, and then connect the dummy source s_0 with all nodes in S and all nodes in $\overline{C}'$ with the dummy sink t_0. The capacities of all arcs outgoing from s_0 and all arcs incident to t_0 are set to infinity.

As the following theorem states, the desired upper bound $U_{out}(S, C)$ can eventually be computed as $1 - \exp(-f^*)$.

Theorem 3.4. *Given an uncertain graph $\mathcal{G} = (N, A, p)$, let $\widehat{G} = (N, A, c)$ be a capacitated graph derived from $\mathcal{G}$ by assigning a capacity $c(a) = -\log(1 - p(a))$ to each arc $a \in A$. Also, given a cluster $C \subseteq N$ and a set of source nodes $S \subseteq C$, let f^* denote the maximum flow from S to $\overline{C}$ on the graph $\widehat{G}$. It holds that $U_{out}(S, C) = 1 - \exp(-f^*)$.*

Proof From the max-flow/min-cut equivalence, it follows that the value f^* of the max-flow is equal to the value c^* of the min-cut. We have

$$\begin{aligned}
f^* &= c^* \\
&= \min_{\mathcal{C}(S,\overline{C})} \sum_{a \in \mathcal{C}(S,\overline{C})} c(a) \\
&= \min_{\mathcal{C}(S,\overline{C})} \sum_{a \in \mathcal{C}(S,\overline{C})} -\log(1-p(a)) \\
&= \min_{\mathcal{C}(S,\overline{C})} -\log \prod_{a \in \mathcal{C}(S,\overline{C})} (1-p(a)) \\
&= -\log \left(\max_{\mathcal{C}(S,\overline{C})} \prod_{a \in \mathcal{C}(S,\overline{C})} (1-p(a)) \right) \\
&= -\log(1-U_{out}(S,C)), \qquad \text{(from Theorem 3.3)}
\end{aligned}$$

which proves the theorem. □

Example 3.2. *Consider the running example in Figures 3.1 and 3.2. The upper bound for the outreach probability from* {s} *to outside cluster* {s,w} *is* 0.80*, due to arcs* (s,w), (s,u)*. It means that the probability that* {s} *reaches any node not belonging to* {s,w} *is lower than or equal to* 0.80*. Similarly, the upper bound for the outreach probability from* {s} *to outside cluster* {s,w,u} *is* 0.496*, due arcs* (u,t), (u,v), (w,v)*. As the probability threshold* $\eta = 0.5$*, all nodes outside cluster* {s,w,u} *can be pruned.*

Finally, it is easy to verify that the properties derived in Theorems 3.1 and 3.2 for outreach probability hold also for the upper bound U_{out}. This result is formulated in Theorem 3.5.

Theorem 3.5. *Given a cluster* $C \subseteq N$ *and a subset* $S \subseteq C$*, the following results hold:*

$$U_{out}(S,C) < \eta \;\Rightarrow\; R(S,t) < \eta, \text{ for all } t \in \overline{C}, \tag{3.4}$$

$$U_{out}(S, C_i) \geq U_{out}(S, C_j), \text{ for all } S \subseteq C_i \subseteq C_j. \tag{3.5}$$

Proof Equation (3.4) follows directly from Theorem 3.1, since $U_{out}(S, C) < \eta \Rightarrow R_{out}(S, C) < \eta \Rightarrow R(S, t) < \eta$, for all $t \in \overline{C}$. Regarding Equation (3.5), we first note that, $C_i \subseteq C_j \Rightarrow \overline{C_i} \supseteq \overline{C_j}$, and, therefore, all cuts from S to $\overline{C_i}$ are also cuts from S to $\overline{C_j}$. This implies that

$$\max_{\mathcal{C}(S,\overline{C_i})} \prod_{a \in \mathcal{C}(S,\overline{C_i})} (1 - p(a)) \leq \max_{\mathcal{C}(S,\overline{C_j})} \prod_{a \in \mathcal{C}(S,\overline{C_j})} (1 - p(a)),$$

or equivalently,

$$\underbrace{1 - \max_{\mathcal{C}(S,\overline{C_i})} \prod_{a \in \mathcal{C}(S,\overline{C_i})} (1 - p(a))}_{U_{out}(S,C_i)} \geq \underbrace{1 - \max_{\mathcal{C}(S,\overline{C_j})} \prod_{a \in \mathcal{C}(S,\overline{C_j})} (1 - p(a))}_{U_{out}(S,C_j)}$$

The theorem follows. □

Single-Source Queries

We next describe how to perform candidate generation when the query set of source nodes is a singleton, i.e., queries are formulated as $RS(\{s\}, \eta)$.

Given a query node s, there exists a single path in the RQ-tree index $\mathcal{T}$ from the leaf cluster $\{s\}$ to the root of $\mathcal{T}$. Our candidate-generation strategy traverses all clusters along this path in a bottom-up fashion i.e., starting from the leaf cluster and going towards the root. The traversal of the path stops as soon as it encounters a *candidate cluster* C^*, whose upper bound $U_{out}(\{s\}, C^*)$ on outreach probability is smaller than η. More formally:

$$C^*(\{s\}, \eta) = \operatorname*{arg\,max}_{\substack{C \supseteq \{s\}, \\ U_{out}(\{s\},C) < \eta}} U_{out}(\{s\}, C).$$

Theorem 3.5 ensures that (i) C^* is the smallest "valid" candidate cluster, i.e., it is the cluster that guarantees that the discarded set $\overline{C^*}$ is as large as possible; and (ii) all nodes $t \notin C^*$ have $R(\{s\}, t) < \eta$, i.e., no true positive is discarded.

Note that, during our bottom-up traversal of $\mathcal{T}$, the upper-bound values $U_{out}(\{s\}, \cdot)$ are computed in a lazy fashion according to the strategy outlined in Algorithm 1. In order to further speed-up query processing, one may consider pre-computing the upper-bound values $U_{out}(\{s\}, C)$, for all clusters $C \in \mathcal{T}$ and all nodes $s \in C$. However, such a pre-computation would lead to increasing the index storage space and, more importantly, the index building time, which could even become unaffordable for large graphs.

Running time. Our candidate generation consists of two steps: the bottom-up traversal of the tree $\mathcal{T}$, and the computation of the upper-bound values U_{out} during that traversal. The first step is linear in the height h of the tree $\mathcal{T}$. The second step requires performing a max-flow computation for each cluster visited during the traversal. As a result, the overall running time of computing the upper-bound values U_{out} is expressed as h max-flow computations. According to Algorithm 1, the max-flow computation depends only on the number of source nodes and the number of arcs outgoing from the source nodes to outside the cluster. Thus, one can upper bound the running time of each max-flow instance by using the size of the subgraph in the last cluster encountered during the traversal. Let $\tilde{n}$ and $\tilde{m}$ denote the number of nodes and arcs in that subgraph. One of the fastest existing max-flow algorithms is the algorithm of Goldberg and Tarjan [69], whose running time is $\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$, where the $\tilde{\mathcal{O}}$ notation hides logarithmic

factors. Assuming that the tree $\mathcal{T}$ is balanced (see Section 3.6), and therefore, $h = \mathcal{O}(\log n)$, the overall time complexity of the candidate-generation phase is still $\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$.

Multiple-Source Queries

In case of queries containing multiple source nodes, one could follow exactly the same candidate-generation strategy as in the single-source case: retrieve the smallest cluster in the index tree that contains all nodes of the query source set S. However, such a strategy may not be very effective in the multiple-source case. The reason is that the cluster enclosing all nodes in S might be a large cluster placed very close to the root of the RQ-tree $\mathcal{T}$. This would affect the efficiency of query processing, as a larger portion of $\mathcal{T}$ would be visited before encountering the desired candidate set, and thus a large number of candidate nodes would need to be verified. Therefore, we discuss next how to select a *set* of clusters (rather than a single cluster common to all source nodes) which may achieve better pruning.

Multiple-source candidate clusters. Our goal is to derive a set of clusters $\{C_i\}_{i=1}^{k}$ of $\mathcal{T}$ whose union set $C_\cup = \bigcup_i C_i$ meets the following requirements: (*i*) all source nodes belong to $C_\cup$; (*ii*) the property of having no false negatives discarded still holds, that is no false negatives are present among the nodes outside $C_\cup$; and (*iii*) the size of $C_\cup$ is minimum, so to guarantee maximum pruning.

Let us translate the above requirements into an optimization problem. To this end, requirements (*i*) and (*iii*) are straightforward to formulate, while for requirement (*ii*) we first

need to derive some theoretical results, which are formally stated in Lemma 3.1 and Theorem 3.6.

Lemma 3.1. *Let $\{C_1, \cdots, C_k\}$ be a set of clusters in $\mathcal{T}$ and $\{S_1, \cdots, S_k\}$ be a set of source node sets, where $S_i \subseteq C_i$, for all i, and $S_i \cap S_j = \emptyset$, for all $i \neq j$. Let also $C_\cup = \bigcup_i C_i$ and $S_\cup = \bigcup_i S_i$. It holds that $U_{out}(S_\cup, C_\cup) \leq 1 - \prod_i (1 - U_{out}(S_i, C_i))$.*

Proof Given any two (disjoint) sets of nodes $X, Y \subseteq N$, let $\mathcal{C}^*(X, Y)$ denote the most-likely cut from X to Y (as defined in Definition 3.2). Let also $\Pr(\neg\mathcal{C}^*(X, Y)) = \prod_{a \in \mathcal{C}^*(X,Y)}(1 - p(a))$ be the probability that $\mathcal{C}^*(X, Y)$ does not exist. First, we note that, by definition, the probability $\Pr(\neg\mathcal{C}^*(X, Y))$ cannot be smaller than the probability that any single valid cut from X to Y does not exist. Given any superset $Y' \supseteq Y$ (such that $X \cap Y' = \emptyset$) it easy to see that $\mathcal{C}^*(X, Y')$ is a valid cut from X to Y too. Thus the following holds:

$$\Pr(\neg\mathcal{C}^*(X, Y)) \geq \Pr(\neg\mathcal{C}^*(X, Y')),$$

for all $Y' \supseteq Y, X \cap Y' = \emptyset$, which implies that

$$\Pr(\neg\mathcal{C}^*(S_i, \overline{C_\cup})) \geq \Pr(\neg\mathcal{C}^*(S_i, \overline{C_i})),$$

as, clearly, $\overline{C_i} \supseteq \overline{C_\cup}$ (and $S_i \cap \overline{C_\cup} = \emptyset$). Furthermore, notice that $\bigcup_i \mathcal{C}^*(S_i, \overline{\mathcal{C}_\cup})$ is a valid cut from $S_\cup$ to $\overline{\mathcal{C}_\cup}$. Hence, based on the same argumentation as above, the following holds:

$$\Pr(\neg\mathcal{C}^*(S_\cup, \overline{C_\cup})) \geq \Pr(\neg\textstyle\bigcup_i \mathcal{C}^*(S_i, \overline{C_\cup})).$$

Finally, it is easy to see that the probability that none of the arcs in the union of multiple cuts exists is lower-bounded by the product of the probability that any single arc in the union cut

does not exist, that is:

$$\Pr(\neg \bigcup_i \mathcal{C}^*(S_i, \overline{C_\cup})) \geq \prod_i Pr(\neg \mathcal{C}^*(S_i, \overline{C_\cup})).$$

In summary, based on the above results, we have:

$$\begin{aligned} \underbrace{\Pr(\neg \mathcal{C}^*(S_\cup, \overline{C_\cup}))}_{1-U_{out}(S_\cup, C_\cup)} &\geq \Pr(\neg \bigcup_i \mathcal{C}^*(S_i, \overline{C_\cup})) \\ &\geq \prod_i Pr(\neg \mathcal{C}^*(S_i, \overline{C_\cup})) \\ &\geq \prod_i \underbrace{Pr(\neg \mathcal{C}^*(S_i, \overline{C_i}))}_{1-U_{out}(S_i, C_i)}, \end{aligned}$$

which implies that

$$U_{out}(S_\cup, C_\cup) \leq 1 - \prod_i (1 - U_{out}(S_i, C_i)).$$

The lemma follows. □

Based on the above lemma, we can now provide the ultimate condition to be ensured for having no false negatives outside $C_\cup$. As formally stated in Theorem 3.6, such a condition is expressed as $1 - \prod_{i \in [1..k]}(1 - U_{out}(C_i \cap S, C_i)) < \eta$.

Theorem 3.6. *Let S be a set of source nodes and $\{C_1, \cdots, C_k\}$ be a set of clusters in $\mathcal{T}$ such that $C_i \cap S \neq \emptyset$, for all i, and $\{C_i \cap S\}_{i=1}^k$ forms a partition of S. Let also $C_\cup$ denote the union set $\bigcup_i C_i$. It holds that:*

$$1 - \prod_i (1 - U_{out}(C_i \cap S, C_i)) < \eta \;\Rightarrow\; R(S, t) < \eta,$$

for all $t \in \overline{C_\cup}$.

Proof For each node $t \in \overline{C_\cup}$, we have

$$R(S,t) \leq R_{out}(S,C_\cup) \leq U_{out}(S,C_\cup) \leq$$

$$1 - \prod_i (1 - U_{out}(C_i \cap S, C_i)) < \eta. \quad \text{(from Lemma 3.1)} \qquad \square$$

The optimization problem we are interested in can now be precisely characterized.

Problem 3.2 (Multiple-source Candidate Generation)**.** *Given an* RQ-tree *index $\mathcal{T}$ and a set of source nodes S, select a set of clusters $\{C_1, \ldots, C_k\}$ of $\mathcal{T}$ so that, for the union set $C_\cup = \bigcup_{i=1}^{k} C_i$, the following holds:*

(i) $S \subseteq C_\cup$*;*

(ii) $1 - \prod_i (1 - U_{out}(C_i \cap S, C_i)) < \eta$*;*

(iii) $|C_\cup|$ *is minimum.*

To solve the above problem, we first discuss a polynomial-time algorithm that provides exact solutions but is not scalable. Then, we derive a more efficient heuristic.

Exact multiple-source candidate generation. The exact solution for Problem 3.2 relies on the dynamic-programming paradigm. Let $\{C_1, \ldots, C_t\}$ be a set of clusters in the RQ-tree $\mathcal{T}$ so that no two clusters in that set are in ancestor–descendant relation. Let also define

$$f(\{C_1, \ldots, C_t\}) = |\bigcup_{i=1}^{t} C_i|, \text{ and}$$

$$\mathcal{U}(S,\{C_1, \ldots, C_t\}) = 1 - \prod_{i=1}^{t} (1 - U_{out}(C_i \cap S, C_i)).$$

For simplicity, assume that $\mathcal{T}$ is binary, and every non-leaf cluster C has two children, right $r(C)$ and left $\ell(C)$. The reasoning can easily be extended to more general trees.

For any cluster C of $\mathcal{T}$ and any integer k, we define $T(C, S, k)$ to be the *minimum* score $\mathcal{U}(S, \{C_1, \ldots, C_t\})$ for a set of clusters $\{C_1, \ldots, C_t\}$ such that

(i) $C_1, \ldots, C_t$ are all descendant of C;

(ii) no two clusters in $\{C_1, \ldots, C_t\}$ are in ancestor–descendant relation;

(iii) for every $s \in S \cap C$, there is (exactly) one cluster in $\{C_1, \ldots, C_t\}$ between s and C; and

(iv) $f(\{C_1, \ldots, C_t\}) \leq k$.

The minimum in the definition of $T(C, S, k)$ is taken over all sets $\{C_1, \ldots, C_t\}$ that satisfy conditions (i)–(iv).

Now, given a query source set S, for every cluster in $\mathcal{T}$ and every $k = 0, \ldots, n$, we compute the scores $T(C, S, k)$ in a bottom-up fashion using the following dynamic-programming equation

$$T(C, S, k) = \min_{k_r + k_\ell \leq k} 1 - (1 - T(r(C), S, k_r))(1 - T(\ell(C), S, k_\ell)).$$

The solution to Problem 3.2 is then given by the smallest value of k for which $T(\text{root}, S, k) < \eta$. This way, only the score of the optimal solution is provided. One can also obtain the solution itself by keeping pointers to the optimal splits $k_r + k_\ell \leq k$ for each C and each k.

It is easy to verify that computing the above exact solution requires $\mathcal{O}(|S| n \log n)$ max-flow computations. Therefore, the exact dynamic-programming algorithm for multiple-source candidate generation may be slow in practice. For this purpose, we introduce next a more efficient greedy heuristic.

Heuristic multiple-source candidate generation. The idea of our heuristic is to perform a number of bottom-up traversals of $\mathcal{T}$ in parallel, one for each $s \in S$. Similar to the single-source case, each traversal proceeds along the path that connects the node s to the root of $\mathcal{T}$. Traversals are performed in a round-robin way and they terminate when the following stopping condition

is met. Let C_i denote the current cluster in $\mathcal{T}$ that encloses node s_i at a certain point of the traversals, for all $s_i \in S$ (note that any two nodes $s_i, s_j \in S$ can be enclosed by the same cluster $C_i = C_j$). Our procedure stops when it reaches the *minimum-sized* union set $C_\cup = \bigcup_i C_i$ for which condition (ii) of Problem 3.2 is satisfied, i.e., $1 - \prod_i (1 - U_{out}(C_i \cap S, C_i)) < \eta$. The final candidate set corresponds to the union set $C_\cup$ of the last clusters reached by the traversal.

Running time. The running time analysis of the (heuristic) multiple-source candidate generation roughly follows the analysis of the single-source case. We need to perform $\mathcal{O}(|S| \log n)$ max-flow computations—contrast to the $\mathcal{O}(|S| n \log n)$ max-flow computations required by the exact method, and $\mathcal{O}(|S| \log n)$ computations of U_{out}. The overall time complexity is therefore $\tilde{\mathcal{O}}(|S| \tilde{n} \tilde{m})$.

3.5.2 Verification

Though guaranteed not to discard any true positive, the candidate set C^* generated according to our candidate-generation strategies may still contain false positive nodes, i.e., nodes t for which $R(S, t) < \eta$. To filter as many of such false positives as possible out of C^*, we propose two alternative verification methods, which are described next. Both verifications take in input the candidate set eventually generated by candidate generation. As a result, there is no difference between single-source verification and multiple-source verification.

Verification based on a lower bound on reliability

The first verification method we propose is based on a lower bound for $R(S,t)$, for any source node set S and a node $t \notin S$. We denote this lower bound by $L_R(S,t)$. The idea is that if we find that $L_R(S,t) \geq \eta$, for some query set S and some node t, then we can safely conclude that t belongs to the solution set.

The lower bound we derive is based on the concept of *most-likely path* from S to t.

Definition 3.3 (most-likely path)**.** *Given a set of nodes S and a node $t \notin S$, the* most-likely path $\mathcal{P}^*(S,t)$ *from S to t is defined as*

$$\mathcal{P}^*(S,t) = \underset{\substack{\mathcal{P} \in \mathbf{P}(s,t),\\ s \in S}}{\arg\max} \prod_{a \in \mathcal{P}} p(a), \tag{3.6}$$

where $\mathbf{P}(s,t)$ denotes the set of all paths from s to t.

The following theorem states that the desired lower bound L_R simply corresponds to the probability of the most-likely path. This probability can be computed by a shortest-path computation on a weighted graph derived from $\mathcal{G}$ by assigning to each arc $a \in A$ a weight $-\log(p(a))$.

Theorem 3.7. *Given a set of source nodes S and a node $t \notin S$, it holds that $R(S,t) \geq L_R(S,t) = \prod_{a \in \mathcal{P}^*(S,t)} p(a)$.*

Proof By definition, $R(S,t)$ is the probability that *at least one path* from a node $s \in S$ to t exists. Hence, $R(S,t)$ is larger than or equal to the probability that *any single path* from some

$s \in S$ to t exists, that is,

$$R(S,t) \geq \prod_{a \in \mathcal{P}} p(a), \text{ for all } s \in S \text{ and all } \mathcal{P} \in \mathbf{P}(s,t).$$

Therefore, we have

$$R(S,t) \geq \max_{\substack{\mathcal{P} \in \mathbf{P}(s,t) \\ s \in S}} \prod_{a \in \mathcal{P}} p(a) = \prod_{a \in \mathcal{P}^*(S,t)} p(a),$$

which proves the theorem. □

Based on the lower bound L_R, the verification step simply consists in keeping only those nodes $t \in C^*$ such that $L_R(S,t) \geq \eta$. This way, the output solution set is guaranteed not to contain any false positive.

An interesting point in this regard would be analyzing the error achieved by the above lower bound. However, such a problem is non-trivial and we leave it for future work. Indeed, one can notice that some extreme cases may arise, that can easily make the error bound analysis hard. One case is given by complete graphs with high probabilities on the arcs, on which large candidate sets tend to be produced, and where our lower-bounding-based verification may be therefore less effective. On the other hand, however, for graphs with small dense substructures that are very commonly encountered in real-world scenarios, the candidate set obtained is typically much smaller. In these cases, as empirically validated in Section 3.7 on several real-world datasets, the lower-bounding-based verification is instead very accurate.

Sampling-based Verification

The above lower-bounding verification guarantees that *no false positive* belongs to the final solution. On the other hand, some false negatives may be introduced. Even though

experimental evidence (Section 3.7) has shown that the recall of our lower-bounding-based verification is always high (within $[0.75, 0.85]$), the accuracy can be further improved.

The idea is to perform sampling (e.g., Monte Carlo) to estimate the reliability of the candidate nodes more accurately than the lower-bounding-based verification. Note that, unlike existing sampling methods discussed in the Introduction [61, 87], sampling here is performed on a small subgraph of the input uncertain graph, that is the subgraph induced by the candidate nodes only. As a result, even though less efficient than the lower-bounding-based verification, this sampling-based verification is still very fast and outperforms the baselines in efficiency, as empirically observed in Section 3.7. One may note that when we sample over the subgraph induced only by the candidate set, we ignore the contribution of the paths passing through nodes not in the candidate set. Since all non-candidate nodes have reliability from the source set less than η, a path from source set to a candidate node that goes though non-candidate nodes also have very small reliability as compared to η, and thereby does not significantly affect the reliability values of candidate nodes. Indeed, the recall of our sampling-based verification is always in the $[0.95, 1.00]$ range. In addition, the number of samples can be used as a knob to tradeoff between efficiency and accuracy.

Running time

For the lower-bound-based verification strategy, one only needs to focus on the subgraph $\tilde{\mathcal{G}}$ of $\mathcal{G}$ induced by the candidate set C^*. This is because our candidate-generation step ensures that all nodes outside the candidate set have reliability from the query source set S less

than η. Therefore, all paths passing through nodes not in the candidate set are guaranteed to have reliability less than η too: this way all nodes (and corresponding arcs) not in C^* can safely be discarded.

According to the reasoning reported in Section 3.5.1, the number of nodes and arcs of $\tilde{\mathcal{G}}$ are upper-bounded by $\tilde{n}$ and $\tilde{m}$, respectively. The lower-bounding-based verification strategy requires to compute the probability of the most-likely path from the source node set S to each node in the candidate set. This can be accomplished with a shortest-path distance computation in $\tilde{\mathcal{G}}$ from the source set S, which leads to a time complexity of $\tilde{\mathcal{O}}(\tilde{m} + \tilde{n})$.

The sampling-based verification, on the other hand, requires to compute all nodes that are reachable from the source set S in every deterministic graph sampled from $\tilde{\mathcal{G}}$. This can be accomplished with a visit of each sampled graph. The total running time is therefore $\mathcal{O}(K(\tilde{m} + \tilde{n}))$, where K denotes the number of samples.

It can be noted that, overall, our query processing ranges from $\tilde{\mathcal{O}}(\tilde{n}\tilde{m})$ time (single-source, lower-bounding-based verification), to $\tilde{\mathcal{O}}(|S|\tilde{n}\tilde{m} + K(\tilde{m} + \tilde{n}))$ time (multiple-source, sampling-based verification). In all cases, however, as $\tilde{n}$ and $\tilde{m}$ are very small in practice (see Section 3.7), the efficiency of our query processing is very high.

3.6 Building the RQ-tree Index

In this section, we provide the guidelines for building the hierarchical structure of the proposed RQ-tree index $\mathcal{T}$. In this regard, we note that:

1. The smaller the height of $\mathcal{T}$, the smaller the size (and therefore the storage space) of $\mathcal{T}$. A small height of $\mathcal{T}$ is achieved when its clusters are partitioned into *balanced* children, i.e., children having roughly the same size.

2. On the other hand, a very small height of $\mathcal{T}$ is also not desirable, as the smaller the height of $\mathcal{T}$, the narrow the range of η values that can be profitably assisted by an RQ-tree index. As an example, think about the extreme case where the height of $\mathcal{T}$ is 1 (which arises when the branching factor of $\mathcal{T}$ is n): such an RQ-tree would be completely useless for our query processing strategy. Within this view, we keep the height of $\mathcal{T}$ of reasonable size by fixing the branching factor of $\mathcal{T}$ to a small number, i.e., 2.

3. Finally, for each cluster C in $\mathcal{T}$, and for each node $s \in C$, we require for $R_{out}(\{s\}, C)$ to be as small as possible, since this would reduce the size of the set produced during candidate generation. As already explained in Section 3.5.1, this improves both efficiency and accuracy of our query processing strategies.

Based on the above requirements, we develop the following method for building an RQ-tree index $\mathcal{T}$. First, according to requirements 1) and 2), we perform a (recursive) *balanced bi-partition* of each non-leaf cluster in $\mathcal{T}$. Requirement 3), instead, provides the basis for the specific criterion to employ for defining each bi-partition. Particularly, for any cluster C in $\mathcal{T}$, the ideal desideratum would be to minimize the single outreach probabilities of each subset $S \subseteq C$, which is clearly unaffordable. Within this view, we first derive an upper bound that is general for the outreach probabilities of all subsets of nodes in a specific cluster, and then we search for the balanced bi-partition that minimizes this upper bound. Note that the upper bound

provided here differs from the U_{out} upper bound derived in Theorem 3.3, because the latter is instead specific for a given set of source nodes. The expression of this general upper bound, denoted by $\mathcal{U}_{out}$, is formalized in the next theorem.

Theorem 3.8. *Given a cluster C, for all sets of source nodes $S \subseteq C$, it holds that* $R_{out}(S, C) \leq \mathcal{U}_{out}(C) = 1 - \prod_{(u,v):u \in C, v \notin C}(1 - p((u, v)))$.

Proof By definition, $1 - R_{out}(S, C)$ corresponds to the probability that no nodes $s \in S$ can reach any node t outside C. In this respect, consider all outgoing arcs of C, i.e., those arcs that connect a node in C with a node outside C. Clearly, the condition that none of these outgoing arcs exists is sufficient to guarantee that no $s \in C$ can get to outside C. Thus, the probability that no outgoing arc of C exists is a lower-bound for $1 - R_{out}(S, C)$. As a result, it holds that $1 - R_{out}(S, C) \geq \prod_{(u,v):u \in C, v \notin C}(1 - p((u, v)))$, or, equivalently, $R_{out}(S, C) \leq 1 - \prod_{(u,v):u \in C, v \notin C}(1 - p((u, v)))$. □

Based on the above reasoning, we can now formalize the optimization problem to be recursively solved for generating the bi-partition of the various clusters in $\mathcal{T}$. The objective is to partition any given cluster $C \in \mathcal{T}$ into two clusters C_1 and C_2 such that (i) $\mathcal{U}_{out}(C_1)$ and $\mathcal{U}_{out}(C_2)$ are simultaneously minimized, and (ii) C_1 and C_2 have roughly the same size. Noticing that minimizing $\mathcal{U}_{out}(\cdot)$ is clearly equivalent to maximizing $1 - \mathcal{U}_{out}(\cdot)$, it is easy to see that these requirements are fully captured by the following optimization problem.

Problem 3.3 (BUILD-RQ-TREE)**.** *Given a cluster $C \in \mathcal{T}$, partition C into two clusters C_1, C_2 such that*

$$\frac{(1-\mathcal{U}_{out}(C_1))(1-\mathcal{U}_{out}(C_2))}{|C_1|} + \frac{(1-\mathcal{U}_{out}(C_1))(1-\mathcal{U}_{out}(C_2))}{|C_2|}$$

is maximized.

As shown in Theorem 3.9, Problem 3.3 is equivalent to MIN-RATIO-CUT [155]. As a result, Problem 3.3 is **NP**-hard.

Theorem 3.9. *Problem 3.3 is* **NP***-hard.*

Proof We prove the theorem by a reduction from MIN-RATIO-CUT. We construct a weighted (deterministic) graph $\widehat{G}$ containing the same set of nodes and arcs as $\mathcal{G}$. We assign to each arc a in $\widehat{G}$ a weight $w(a) = -\log(1-p(a))$, and we make $\widehat{G}$ undirected by simply ignoring the directness of each arc. Given any two node sets $N_i, N_j \subseteq N$, let $A(N_i, N_j)$ denote the set of all arcs in $\widehat{G}$ between N_i to N_j. Solving MIN-RATIO-CUT on $\widehat{\mathcal{G}}$ finds a bi-partition $\{N_1, N_2\}$ of the node set N that minimizes $\frac{1}{|N_1|}\sum_{a\in A(N_1,N_2)} w(a) + \frac{1}{|N_2|}\sum_{a\in A(N_1,N_2)} w(a)$, or, equivalently, that maximizes $\frac{1}{|N_1|}\prod_{a\in A(N_1,N_2)}(1-p(a)) + \frac{1}{|N_2|}\prod_{a\in A(N_1,N_2)}(1-p(a)) = \frac{1}{|N_1|}(1-\mathcal{U}_{out}(N_1))(1-\mathcal{U}_{out}(N_2)) + \frac{1}{|N_2|}(1-\mathcal{U}_{out}(N_1))(1-\mathcal{U}_{out}(N_2))$. The theorem follows. □

The similarity with MIN-RATIO-CUT implicitly provides us with a well-founded approach for heuristically solving our BUILD-RQ-TREE problem. Indeed, one can resort to one of the various well-established heuristics for MIN-RATIO-CUT defined in the literature. Specifi-

Algorithm 2 BuildRQtree

Input: an uncertain graph $\mathcal{G} = (N, A, p)$

Output: an RQ-tree index $\mathcal{T}$

1: $\mathbf{C} \leftarrow \{N\},\ \mathcal{T} \leftarrow \{\mathbf{C}\}$

2: **repeat**

3: $\quad \mathbf{C}' \leftarrow \emptyset$

4: $\quad$ **for all** $C \in \mathbf{C}$ s.t. $|C| > 1$ **do**

5: $\quad\quad$ build $\widehat{G} = (\widehat{N}, \widehat{A}, w)$, where $\widehat{N} = C$, $\widehat{A} = \{(u, v) \mid (u, v) \in A, u \in C, v \in C\}$, and $w(a) = -\log(1 - p(a))$, for all $a \in \widehat{A}$

6: $\quad\quad \{C_1, C_2\} \leftarrow$ METIS $(\widehat{G})$

7: $\quad\quad \mathbf{C}' \leftarrow \mathbf{C}' \cup \{C_1, C_2\}$

8: $\quad$ **end for**

9: $\quad \mathbf{C} \leftarrow \mathbf{C}',\ \mathcal{T} \leftarrow \mathcal{T} \cup \{\mathbf{C}\}$

10: **until** $\mathbf{C} = \emptyset$

cally, in our framework, we use the METIS algorithm [90], as a good trade-off between accuracy and efficiency.

All steps of our strategy for building an RQ-tree are summarized in Algorithm 2.

Index building time. Given a cluster C in $\mathcal{T}$, let n_C and m_C denote the number of nodes and arcs in the subgraph of the input uncertain graph $\mathcal{G}$ identified by the nodes in C, respectively. Computing a bi-partition of C by means of the METIS algorithm takes $\mathcal{O}(n_C + m_C)$ time. Running METIS on all clusters of any single level of $\mathcal{T}$ takes $\mathcal{O}(\sum_C (n_C + m_C))$. As all

clusters in any single level of $\mathcal{T}$ forms a partition of the whole set of nodes in $\mathcal{G}$, the latter is equivalent to $\mathcal{O}(n+m)$. The number of levels (height) of $\mathcal{T}$ is $\mathcal{O}(\log n)$, as our RQ-tree index building strategy guarantees for $\mathcal{T}$ to be a balanced tree. As a result, the overall time complexity of building an RQ-tree index is $\mathcal{O}((n+m)\log n)$.

Index storage space. As explained above, the height of $\mathcal{T}$ is $\mathcal{O}(\log n)$. Each level of $\mathcal{T}$ contains a partition of the whole set of nodes in $\mathcal{G}$, thus each node in $\mathcal{G}$ is stored exactly one time for each level. Hence, the overall storage space required by an RQ-tree index is $\mathcal{O}(n \log n)$.

3.7 Experimental Results

In this section, we present our empirical analysis. We are mainly aimed at assessing:

- Efficiency and effectiveness of the proposed RQ-tree index in terms of recall, query-processing time, index-building time, and index size (Section 3.7.2); we also compare our approach with two baselines, namely Monte-Carlo sampling (MC-sampling [61]), and an alternative sampling technique designed for source-to-target reliability (RHT-sampling [87]).
- Effectiveness of our approach by breaking-down the analysis to the two phases of our query processing, i.e., candidate generation and verification (Section 3.7.3).
- Performance with varying the size of the query source set (Section 3.7.4).

- Scalability of the proposed method with respect to query processing, index building time, and index size (Section 3.7.5).

Furthermore, as an example of application, we show in Section 3.7.6 how our RQ-tree index can help speeding-up the iterative hill-climbing greedy algorithm [92] for the *influence-maximization* problem.

All the code is implemented in C++ and experiments are performed on a single core of a 100GB, 2.50GHz Xeon server.

3.7.1 Experiments Settings

Datasets. We involve six real-world datasets, each representing a directed uncertain graph.

DBLP [47]. We consider the usual co-authorship graph where an arc connects two authors if they co-authored at least once. We make the graph directed by considering arcs in both directions. We derive arc probabilities as described in [87, 131]: we consider an exponential cdf of mean μ to the number of collaborations; hence, if two authors collaborated c times, we compute the corresponding probability as $1 - \exp^{-c/\mu}$. We consider $\mu \in \{2, 5, 10\}$ in our experiments. Keeping fixed the collaborations, higher values of μ generate smaller probabilities (see Figure 3.3).

Flickr [62]. Flickr is a popular online community, where users share photos, participate in common-interest groups, and form friendships. We create a graph from a recent snapshot of Flickr, by linking two users if they belong to at least one common interest group. Like DBLP, the undirected graph is made directed by simply considering the arcs in both directions. Following

Table 3.2: Reliability Queries: Dataset Characteristics.

Dataset	# Nodes	# Arcs
DBLP	$684,911$	$4,569,982$
Flickr	$78,322$	$20,343,018$
BioMine	$1,008,201$	$13,445,048$
WebGraph	$10,000,000$	$174,918,788$
Last.FM	$6,899$	$24,144$
NetHEPT	$15,235$	$62,776$

[131], we assign probability to an arc by computing the Jaccard coefficient of the groups that the two users belong to.

BioMine. This is a recent snapshot of database of the BIOMINE project [136], which is a collection of biological interactions. The graph is directed, and with probability associated to the arcs [136].

WebGraph [154]. This is the uk-2007-05 web graph data [27]. For our experiments, we use a subset containing 10M pages and 175M hyperlinks. For a directed arc (u, v), the probability is computed as $\frac{1}{d(u)}$, where $d(u)$ is the out-degree of node u [71].

Last.FM [105]. Last.FM is a music web site, where users listen to their favorite tracks, and communicate with each other based on their music preferences. We crawled a local network of Last.FM, and formed a directed graph by connecting two users if they communicated at least once. Like WebGraph, the probability of a directed arc (u, v) is $\frac{1}{d(u)}$.

NetHEPT [122]. This graph is created from the "High Energy Physics - Theory" section of the e-print arXiv with papers from 1991 to 2003. Like DBLP, two authors are connected by directed arcs if they co-authored at least once. In this graph, we follow [38] and assign constant arc probabilities (0.5).

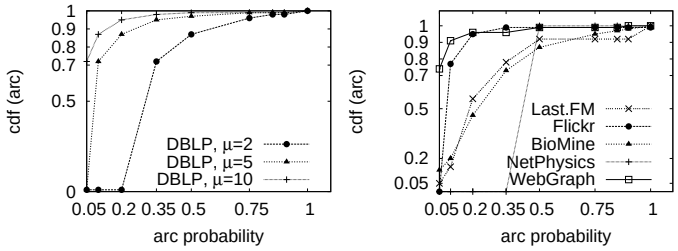


Figure 3.3: Reliability Queries: Cumulative Distribution of Arc Probabilities.

Table 3.3: Comparison between RQ-tree and Baselines: Recall.

	Last.FM			*NetHEPT*			
η	RHT	RQ-tree-MC	RQ-tree-LB	MC	RHT	RQ-tree-MC	RQ-tree-LB
0.4	0.97	1	0.95	1	0.95	0.98	0.78
0.6	0.98	1	0.97	1	0.96	0.98	0.82
0.8	0.98	1	0.97	1	0.96	1	0.88

The main characteristics of the selected datasets are reported in Table 3.2 (sizes) and Figure 3.3 (cumulative distribution of arc probabilities). We use the first three datasets for assessing index performance, while *WebGraph* (the largest one) is used for assessing scalability. The last (smaller) two datasets are used for comparison with the baselines, both in terms of general performance and in the influence-maximization application. Involving smaller datasets for these comparisons is needed to allow the various baselines to terminate in reasonable time.

Query workload. For single-source queries, we select a node uniformly at random. For multiple-source queries, we select uniformly at random a set of nodes from a subgraph of the original graph. We vary the cardinality of the query set from 2 to 20, and the diameter of the subgraph from 2 to 6. Results are averaged over 100 sets of nodes, while the probability threshold η is varied from 0.4 to 0.8.

Table 3.4: Comparison between RQ-tree and Baselines: Query-Processing Time (sec).

	Last.FM				*NetHEPT*			
η	MC	RHT	RQ-tree-MC	RQ-tree-LB	MC	RHT	RQ-tree-MC	RQ-tree-LB
0.4	16.5	6.21	0.1	0.008	27.23	2353	15.97	0.010
0.6	16.5	6.21	0.08	0.007	27.23	2353	15.96	0.008
0.8	16.5	6.21	0.08	0.006	27.23	2353	15.64	0.006

Competing methods. We evaluate the performance of our RQ-tree by focusing on both the verification strategies proposed in Section 3.5.2. Particularly, we hereinafter denote by RQ-tree-LB the variant involving lower-bounding-based verification, and by RQ-tree-MC the variant involving (Monte-Carlo-)sampling-based verification. We compare both RQ-tree-LB and RQ-tree-MC with the following baselines:

MC-Sampling. We consider a basic Monte-Carlo-sampling method [61] running on the whole graph. We derive K deterministic graphs by sampling the given uncertain graph according to the arc probabilities. Then, we compute the set of all nodes reachable from the query node(s) in each sampled graph. Eventually, all nodes that are reachable from the query in at least ηK sampled graphs form the reliable set. Such a baseline method has time complexity $O(K(m+n))$.

RHT-Sampling. This is a method proposed in [87] as a fast alternative to Monte-Carlo sampling for the source-to-target reliability problem. The method is designed for reliability between a single pair of nodes; thus, in order to compute the whole reliable set, it needs to be run n times, every time using a different node in the graph as target node. As stated in [87], the time complexity of a single execution of RHT-Sampling is $\mathcal{O}(nd)$ (where d is the diameter of the graph), hence for reliable set computation the complexity becomes $\mathcal{O}(n^2 d)$. For this purpose,

even being faster than Monte-Carlo sampling for source-to-target reliability, we do not expect high efficiency for this baseline in solving our generalized problem.

For all sampling-based methods, i.e., the two baselines and our RQ-tree-MC, we have observed accuracy convergence on all datasets with a number of samples K around $1,000$. This is roughly the same number observed in [87, 131]. Hence, we set $K = 1,000$ for all sampling-based methods, in all experiments.

Table 3.5: RQ-tree: Recall over Various Datasets (Single-Source Queries).

	RQ-tree-MC			RQ-tree-LB		
	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$
DBLP	0.99	0.99	1.00	0.75	0.87	0.91
Flickr	0.98	0.99	0.99	0.76	0.79	0.83
BioMine	0.97	0.98	0.98	0.77	0.81	0.85

Table 3.6: RQ-tree: Query-Processing Time (sec) over Various Datasets (Single-Source).

	RQ-tree-MC			RQ-tree-LB			MC
	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	all η
DBLP	43.01	40.48	36.83	1.50	0.60	0.60	3081.85
Flickr	60.23	58.60	54.75	0.21	0.20	0.17	5058.55
BioMine	6061.90	5416.84	4974.09	1.00	0.50	0.50	25608.40

Table 3.7: RQ-tree: Recall w/ Varying Arc Probabilities (DBLP, Single-Source Queries).

	RQ-tree-MC			RQ-tree-LB		
	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$
$\mu = 2$	0.95	0.96	0.98	0.52	0.75	0.76
$\mu = 5$	0.96	0.97	0.98	0.75	0.87	0.91
$\mu = 10$	0.97	0.97	0.99	0.89	0.91	0.96

Table 3.8: RQ-tree: Query Processing Time (sec) w/ Varying Arc Probabilities (DBLP, Single-Source Queries).

	RQ-tree-MC			RQ-tree-LB			MC
	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	$\eta = 0.4$	$\eta = 0.6$	$\eta = 0.8$	all η
$\mu = 2$	152.94	145.72	141.80	2.50	0.82	0.68	11476.60
$\mu = 5$	43.01	40.48	36.83	1.50	0.60	0.57	3081.85
$\mu = 10$	38.70	36.15	33.10	1.40	0.57	0.57	2257.55

3.7.2 General Performance

Comparison with baselines. We compare our RQ-tree-LB and RQ-tree-MC with MC-Sampling and RHT-Sampling baselines. As said, for this comparison we focus on the two smaller datasets only (i.e., *Last.FM* and *NetHEPT*) to allow the baselines to terminate in reasonable time.

Computing the exact reliable set is computationally unfeasible. For this purpose, to measure accuracy, we use the reliable set returned by MC-sampling as a proxy. Particularly, we are interested in assessing accuracy in terms of *recall*. Denoting by T the reliable set outputted by any selected method and by T^* the reliable set produced by MC-Sampling, we define recall as $\frac{|T \cap T^*|}{|T^*|}$. This way, the recall of MC-sampling is clearly *always 1*, thus we avoid to report it. Note that *precision* provides instead less interesting evidence of the accuracy of our methods, then we leave it out of the presentation. Indeed, our RQ-tree-LB guarantees perfect precision (no false positives), while, as far as RQ-tree-MC, even though it might return false positives, its verification strategy is more accurate than RQ-tree-LB, and thus it guarantees very high precision as well.

We report recall and query-processing time of the selected methods (with varying η) in Table 3.3 and 3.4, respectively. Table 3.3 shows that our RQ-tree-MC achieves the best recall results among all methods on both *Last.FM* and *NetHEPT*, being also more accurate than the

RHT baseline on all datasets and for all η. Particularly, it achieves *perfect* recall for all η on *Last.FM* and for $\eta = 0.8$ on *NetHEPT*. As far as RQ-tree-LC, it is in general less accurate than RQ-tree-MC, as expected. However, its recall is very close to 1 (0.95-0.97) and comparable to RHT on *Last.FM*. On the *NetHEPT* dataset, the recall of RQ-tree-LC decreases, even though it is on average greater than 0.82. A possible justification for this is given by the higher arc probabilities present on *NetHEPT* as compared to those of *Last.FM*.

Regarding efficiency (Table 3.4), our RQ-tree-LB and RQ-tree-MC drastically outperform the baselines. RQ-tree-MC is up to 2 and 3 order of magnitude faster than RHT-Sampling and MC-Sampling, respectively. RQ-tree-LB is even much better: it outperforms RHT-Sampling and MC-Sampling up to 6 and 4 orders of magnitude, respectively. We also note that, on the smaller *Last.FM* dataset, RHT is more efficient than MC-Sampling, while on *NetHEPT* the opposite happens. This is expected, as, when the size of the graph increases, RHT becomes very expensive due to its time complexity quadratic in the number of nodes in the graph. For instance, on larger graphs such as *BioMine* and *Flickr*, RHT could not finish in one day. Therefore, in the remainder of this section, we compare the efficiency of our methods with MC-Sampling only.

Index building and query processing. We now shift the attention on larger datasets, i.e., *DBLP*, *Flickr*, and *BioMine*. We first report the basic statistics about the RQ-tree index in Table 3.9, where it can be observed that the offline index building time is quite modest for all datasets: for instance, building the index on *BioMine* (1M nodes and 13M arcs) takes about 50 minutes. The space requirement is contained as well: on *BioMine* our index takes approximately

Table 3.9: RQ-tree: Index Building Time, Index Size, Height of the Tree, and Number of Clusters.

Dataset	Time (sec)	Size (MB)	Height	# Clusters
DBLP ($\mu = 5$)	1,855	123	14	735,424
Flickr	1,649	118	11	80,726
BioMine	2,890	203	15	1,040,750

200 MB. In Section 3.7.5, we report a deeper analysis of how index building time and space requirements scale as the graph size grows.

In Table 3.5 and 3.6 we show recall and query-processing time of our RQ-tree-LB and RQ-tree-MC, and the MC-Sampling baseline. The recall of RQ-tree-MC is always close to 1 (never less than 0.97), while the recall of RQ-tree-LB is reasonably high as well: it is 0.82 on average, up to 0.91, and never less than 0.75. In general, the recall of RQ-tree-LB increases as η increases. This is due to the lower-bounding verification method, which is based on the most-likely path between source and target nodes: higher probability thresholds leads to tighter lower bounds.

As far as query-processing times (Table 3.6), both our methods are evidently much efficient than the MC-Sampling baseline. RQ-tree-MC is 1 order (*BioMine*) or 2 orders (*DBLP* and *Flickr*) of magnitude faster than the baseline, while the speed-up achieved by RQ-tree-LB is 3–5 orders of magnitude. We also note that the runtimes decrease as η increases. This is because a higher threshold leads to better chance of having a smaller candidate set, which reduces the time spent in verification.

Performance with varying arc probabilities. We next analyze the performance of RQ-tree with varying arc probability values, while keeping the structure of the graph fixed. Table 3.7

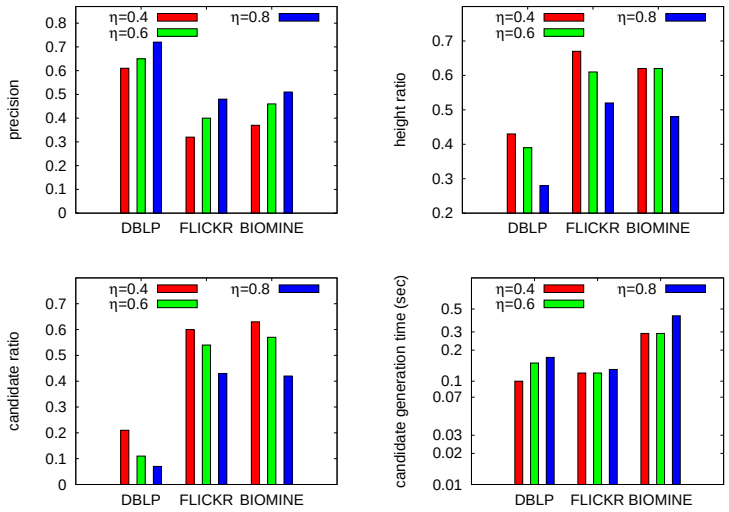


Figure 3.4: RQ-tree Candidate Generation: Results on DBLP, Flickr, Biomine

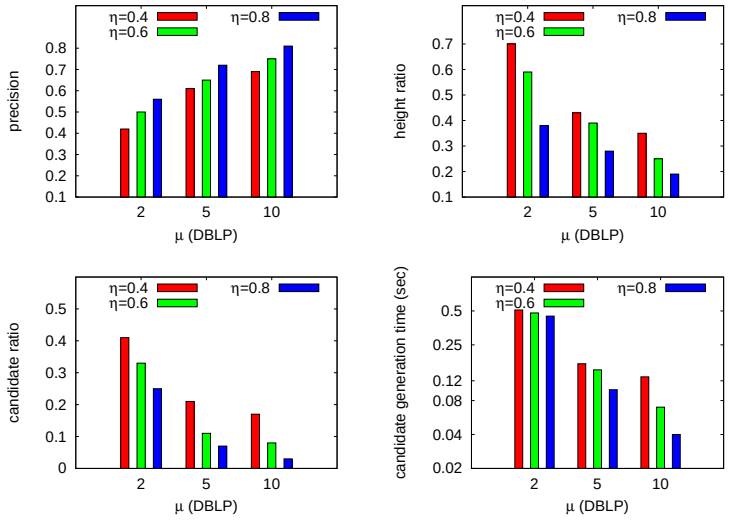


Figure 3.5: RQ-tree Candidate Generation: Results on DBLP with Varying Arc Probabilities

and 3.8 report query performance over *DBLP*, where the arc probabilities are assigned with $\mu = 2, 5$ and 10, respectively (higher values of μ generate smaller arc probabilities). The recall of RQ-tree-MC is always greater than 0.97 and it does not depend a lot on arc probabilities.

The recall of RQ-tree-LB instead is clearly increasing as arc probabilities decrease. This is due to the RQ-tree-LB verification method, which considers the most-likely path between source and target nodes as a lower bound, and the smaller the probabilities the tighter the lower bound. Both our RQ-tree methods and MC-Sampling have improved efficiency with smaller arc probabilities. The efficiency of RQ-tree-MC and MC-Sampling improves because the smaller the arc probabilities, the smaller the number of arcs in the various sampled graphs. However, also RQ-tree-LB gets faster as the arc probabilities get smaller, because this implies a smaller-sized candidate set.

3.7.3 Insights into the Candidate-Generation Phase

We next focus on the RQ-treecandidate-generation phase, which is common to both our RQ-tree-LB and RQ-tree-MC. We report the following measurements:

- **Precision:** defined as $\frac{|T \cap T^*|}{|T|}$, where T is the set produced by RQ-tree candidate generation and T^* is the MC-Sampling reliable set;
- **Height Ratio:** defined as the ratio of the height of the RQ-tree traversed during candidate generation over the total height of the RQ-tree;
- **Candidate Ratio:** defined as the ratio of the RQ-tree candidate-set size, over the total number of nodes in the uncertain graph. This provides an indication of the pruning power of the RQ-tree candidate generation.

The above measurements are reported in Figure 3.4 for the *DBLP* (with $\mu = 5$), *Flickr*, and *BioMine* datasets, and in Figure 3.5 for *DBLP* with varying arc probabilities.

As expected, both the height traversed in the RQ-tree and the size of the candidate set returned by our candidate generation decrease with higher η. Specifically, in *DBLP*, for $\eta = 0.8$, the height traversed is $1/4$ of the entire index tree height, while the size of the candidate set is only 7% of the total nodes. The precision of the candidate generation phase improves as the probability threshold increases, e.g., precision is 0.75 for $\eta = 0.8$ in *DBLP*. However, in many cases the precision is around (or even below) 0.5, meaning that half of the candidates are not part of the final solution, thus confirming the need for verification. We also observe that the performance of our index improves with smaller arc probabilities. This is because the probability that a node can get to outside its cluster decreases, thus strengthening the pruning power of RQ-tree.

Figures 3.4 and 3.5 also report the candidate-generation running times, which are decreasing with smaller arc probabilities and larger probability thresholds.

3.7.4 Performance with Varying Source-Set Size

We next analyze the performance of the RQ-tree index for multiple-source queries. For the sake of brevity, here we focus only on the RQ-tree-LB variant. Table 3.10 reports query-processing results on *DBLP* with $\mu = 5$ and $\eta = 0.6$. The table reports recall of our overall query-processing method, precision of the candidate generation phase, height ratio, and query-processing time. We vary both query set size (2, 5, 10, and 20) and query diameter d, i.e., the diameter of the subgraph (of the original uncertain graph) from which the queries are

Table 3.10: RQ-tree-LB Query-Processing Results on DBLP ($\mu = 5$, $\eta = 0.6$), Varying the Size of the Set of Query Nodes (1^{st} Column) and the Diameter (d) of the Subgraph from which these Nodes were Picked.

Recall of the Overall Method, Precision of the Candidate Generation Phase, Height Ratio, and Query-Evaluation Time.

# nodes	recall			precision			height ratio		
	$d=2$	$d=4$	$d=6$	$d=2$	$d=4$	$d=6$	$d=2$	$d=4$	$d=6$
2	0.85	0.86	0.82	0.65	0.61	0.55	0.40	0.41	0.44
5	0.82	0.85	0.82	0.60	0.45	0.24	0.40	0.57	0.81
10	0.82	0.81	0.81	0.55	0.37	0.17	0.40	0.80	0.87
20	0.76	0.76	0.75	0.55	0.17	0.13	0.45	0.93	0.95

# nodes	RQ-tree-LB runtime (sec)			MC runtime (sec)		
	$d=2$	$d=4$	$d=6$	$d=2$	$d=4$	$d=6$
2	0.60	0.60	0.67	8502	8431	8130
5	0.61	0.87	2.50	10500	10500	11352
10	0.60	2.35	3.32	11200	11200	11200
20	0.71	3.41	4.20	13600	14900	15100

Table 3.11: Scalability Analysis using Single-Source Queries with $\eta = 0.6$ on the WebGraph dataset.

# Nodes and # Arcs	RQ-tree characteristics			RQ-tree-LB runtime (sec)	
	Size	Height	# Clusters	Index building	Query processing
1M, 15M	62MB	17	1,202,754	1,221	0.11
3M, 50M	177MB	18	3,410,221	7,312	0.13
5M, 81M	421MB	19	5,810,934	11,273	0.17
7M, 122M	813MB	21	9,570,259	25,315	0.21
10M, 175M	1,220MB	21	11,758,022	37,146	0.27

randomly selected ($d = 2, 4$, and 6). Note that, as the query diameter increases, it is more likely that the smallest cluster containing all the query nodes is close to the root of the RQ-tree.

We observe the following. (1) The recall of our method is always in the range $[0.75, 0.85]$, which confirms the high effectiveness of our verification method. (2) The effi-

ciency is clearly decreasing as the size of the source set increases. However, the speed-up with respect to MC-Sampling is still very significant: at least 4 orders of magnitude.

3.7.5 Scalability

We analyze the scalability of our RQ-tree on the *WebGraph* dataset. For these experiments, we consider subgraphs of the original *WebGraph* with a number of nodes 1M, 3M, 5M, 7M, and 10M, respectively. The corresponding index building space and time, as well as the query processing time, are reported in Table 3.11. We observe that the index time increases polynomially with the number of nodes in the uncertain graph, while the search time is linear with respect to the graph size. The results assess the high scalability of our RQ-tree.

3.7.6 Application: Influence Maximization

The problem known as *influence maximization* [92], whose most significative application is *viral marketing*, has received a great deal of attention in the data mining literature over the last decade. It requires to find a set S of cardinality k such that it maximizes the *expected spread*, i.e., the expected number of nodes that would be infected by a viral cascade started in S, according to an underlying propagation model. A popular propagation model for influence maximization is the *independent cascade model* [92]: each active neighbor v of a node u has one shot at influencing u and succeeds with the probability $p(v, u)$ associated to the arc (v, u), which represents the strength of the influence of v on u. Given a directed probabilistic graph

$\mathcal{G} = (N, A, p)$, the expected spread $\sigma(S)$ of a set of nodes $S \subseteq A$ is

$$\sigma(S) = \sum_{G \sqsubseteq \mathcal{G}} \Pr(G) \sum_{t \in A} P_G(S, t).$$

Rearranging the terms and using the notation reported in Equation (3.2), the expected spread can be rewritten as

$$\sigma(S) = \sum_{t \in A} R(S, t).$$

The problem of finding the set S of cardinality k that maximizes $\sigma(S)$ is **NP**-hard. However, thanks to the submodularity and monotonicity of $\sigma(S)$, the simple Greedy approach that iteratively adds to S the node that brings the largest marginal gain in the objective function, provides $(1 - 1/e)$ approximation guarantee [92]. Unfortunately, finding such a node requires to compute reliability, which is **#P**-complete. Therefore, the existing body of research usually applies sampling methods (e.g., Monte Carlo) to compute the best seed node at each iteration of the Greedy algorithm.

We next show how the classic Greedy algorithm can exploit an RQ-tree index to avoid resorting to costly Monte-Carlo sampling, thus achieving very high speed-up and paying almost nothing in terms of accuracy.

At each iteration of the Greedy algorithm, given the current set of selected nodes S, we need to compute the node $w \in N \setminus S$ such that $\sum_{t \in A} R(S \cup \{w\}, t)$ is maximum.

We use a histogram-based method to exploit the RQ-tree index. We fix a few probability threshold values in ascending order, i.e., $\eta_1 < \eta_2 < \ldots < \eta_p$. Let $f(S, \eta_i)$ denote the size of the reliable set $RS(S, \eta_i)$: we determined the expected spread from S as $f(S, \eta_p)\eta_p + [f(S, \eta_p) - f(S, \eta_{p-1})]\eta_{p-1} + \ldots + [f(S, \eta_2) - f(S, \eta_1)]\eta_1$.

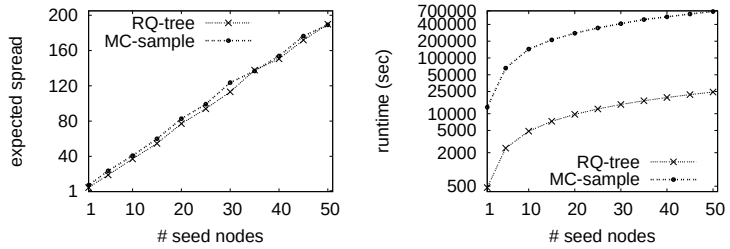


Figure 3.6: Influence Maximization: Last.FM

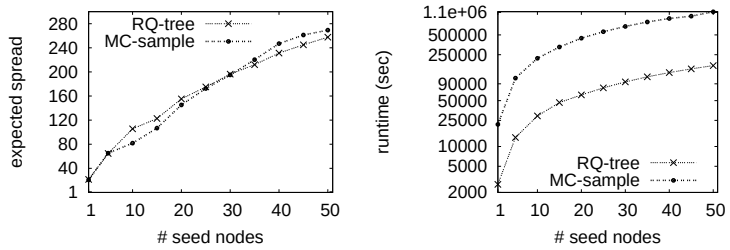


Figure 3.7: Influence Maximization: NetHEPT

Next, we compare the standard Greedy algorithm coupled with Monte-Carlo sampling (using 1,000 samples), and the same algorithm empowered with the RQ-tree index: the results on *Last.FM* and *NetHEPT* are reported in Figure 3.6 and 3.7, respectively.

We measure the effectiveness of the methods as the expected spread achieved by the set of nodes selected. This is computed via Monte-Carlo sampling. We observe that the expected spread achieved by the two methods (left plots in Figure 3.6 and 3.7) is almost the same, while, as far as running time, employing the RQ-tree index allows at least one order of magnitude of speed-up (right plots in Figure 3.6 and 3.7). For instance, RQ-tree requires 7 hours to identify a seed set of 50 nodes in *Last.FM*, as compared to 8 days using Greedy along with Monte-Carlo sampling.

3.8 Summary

In this chapter, we introduce RQ-tree, a novel index to efficiently answer a generalized version of reliability queries in uncertain graphs. The proposed index is based on a hierarchical partitioning of the nodes of the graph above which we apply a candidate generation phase based on maximum flow, and a verification phase based on either a lower-bounding strategy or sampling.

Our experimental results over several real-world datasets show that: (1) RQ-tree is up to six orders of magnitude faster than the state-of-the-art sampling methods, while guaranteeing high accuracy in terms of recall ($[0.75 - 0.98]$); (2) The effectiveness of our approach improves with smaller arc probabilities, and also with higher probability thresholds; (3) RQ-tree scales very well with the size of the input graphs. Finally, we also show how the use of RQ-tree index in the well-known influence-maximization problem obtains a large speed-up over state-of-the-art methods, while achieving comparable accuracy.

In future work, we plan to extend our method for reliability queries when the arc probabilities are not independent. We also plan to apply our RQ-tree in several diverse applications.

Chapter 4

Social Influence Maximization

"Is Social Intelligence More Useful than IQ?"

Daniel Goleman, author of the *'Social Intelligence'*

A central characteristic of social networks is that it facilitates rapid dissemination of information between large groups of individuals. This chapter will examine the problem of determination of *information flow representatives*, a small group of authoritative representatives to whom the dissemination of a piece of information leads to the maximum spread. Clearly, information flow is affected by a number of different structural factors such as the node degree, connectivity, intensity of information flow interaction and the global structural behavior of the underlying network. We will propose a stochastic information flow model, and use it to determine the authoritative representatives in the underlying social network. We will first design a heuristic, but more accurate RankedReplace algorithm, and then use a Bayes probabilistic model in order to approximate the effectiveness of this algorithm with the use of a fast algorithm. We will

examine the results on a number of real social network data sets, and show that the method is more effective than state-of-the-art methods.

4.1 Introduction

In recent years, social networks have found increasing popularity because of their ability to connect geographically disparate groups of individuals. Social networks are well known to enjoy the benefits of the *network effect*, wherein the increase in the size of the social network also increases the perceived benefits of using it. Much of this benefit is embedded in the information flows in the social network. These information flows arise as a result of the communication between the different entities in the social network. The information flow is also impacted by the network topology and the intensity of information flow interactions between different nodes. Since information flows play such a key role in the popularity of social networks, significant research has been performed in recent years to characterize important characteristics of such flows [108, 109].

A key question which arises in the context of social networks is to determine the *information flow authorities* in the social network. Information flow authorities are defined as a very small group of members at which the dissemination of information leads to the most rapid spread throughout the social network. The concept of information authorities is peripherally related to that of the concept of *hubs and authorities* in web networks [99]. The concept of hubs and authorities is used in order to find central points of influence in web networks. However, the concept of *information flow authorities* is quite different from that of the hub-authority frame-

work, in that it is more critically dependent upon the structure of the *flows along the underlying network*. This is dependent both upon the structural characteristics of the network and the flow intensity along different edges (which can be measured in many applications). In the experimental section, we will see that the use of a purely structural method (called *PeerInfluence*) is not sufficient for effective determination of flow authorities. The use of an *information flow model* is critical in determining the best nodes for information dissemination. Furthermore, our model also allows for the development of particular variants which can target specific nodes for influence. This is interesting in a number of applications in which only a subset of the nodes may be relevant for dissemination of information.

Clearly, the flow authorities in the social network are likely to be central and well connected entities in the network. This is related to the concept of determining *central nodes* [65] in graphs and social networks. However, the local structural measures alone do not provide a global view of the *centrality of flows* in the social network. Rather, the flow centrality is defined by the global topology, and the pattern of interactions between different nodes. A related problem is that of virus propagation in computer networks and epidemic spreading [34, 123]. It has been observed in earlier work [34], that the flow of information in social networks, blogs, and network-based product-recommendation systems is very similar to that of virus spread in computer networks. It has been observed in this work that the structure of the network and the interaction intensities between nodes can play a critical role in the information dissemination process.

We will design a stochastic approach in order to model the flow behavior in social networks. We will leverage this flow model in order to design an approach (called RankedReplace) for determining flow authorities in social networks. Then, we will approximate the flow model with a random-walk based model in conjunction with a probabilistic Bayes algorithm. We will refer to this algorithm as the BayesTraceback algorithm. This approximation is very efficient, and turns out to be almost equally effective in practice. We will show that our techniques are much more effective than state-of-the-art techniques which can be adapted to this problem.

4.2 Related Work

Social networks represent individuals and their relationships, such as friendships, collaborations, or recommendation seeking relations. There are dedicated websites, such as Facebook [53], Twitter [146], Orkut [125], hi5 [77], Myspace [121], LiveJournal [112], Last.FM [105], and Delicious [49] which provide online social networking capabilities. Social networks have been shown to have advantages as a medium for fast, widespread information cascade. They provide rapid access to large scale news data, sometimes even faster than the mass media, e.g. the announcement of death of Michael Jackson [1]. They can serve as a medium to collectively achieve a social goal. For instance, with the use of group and event pages in Facebook and Twitter, events such as "2011 Egyptian Protest" quickly reached to the protestors worldwide [132]. Social networks can also perform as a platform for online marketing [52, 70, 113]. Thus, it is important to find a small subset of influential individuals who can influence the largest number of people in a social network. More formally, the *influence maximization problem* can

be stated as follow: Given a probabilistic model for influence, determine a set of k seed nodes generating the largest expected information cascade. The formulation of influence maximization as an optimization problem is due to Domingos and Richardson [52], who modeled influence by an arbitrary Markov random field, and provide heuristics for maximization. The first provable approximation guarantees are given by Kempe, Kleinberg, and Tardos in [92,93]. Several heuristics have been also proposed to improve the efficiency of that method [37,38,71,108].

Recently, there have been several works on graph influence maximization in the presence of a competing negative information spread [25, 32, 36, 111]. In [104], Lappas et. al. introduced the concept of k-effectors. The k-effectors problem identifies k seed nodes, such that, the spread of an information is maximized over a set of given nodes and minimized outside the set.

4.3 Flow Authority Model for Social Networks

In this section, we will introduce the flow authority model for social networks. We assume that the universal set of nodes over which the social network is defined is denoted by U, and the edge set by A. Therefore, the underlying graph is denoted by (U, A). The graph is assumed to be directed, since information flows are specific to direction in the most general case. However, this assumption is not specific to the techniques discussed in this work and they can easily be applied to undirected networks. This can be achieved by replacing an undirected edge with two symmetric directed edges. The set of nodes from which an incoming edge is incident into node i is denoted by $N(i)$. In other words, we have $N(i) = \{k : (k, i) \in A\}$. The

set of nodes on which the outgoing edges of i are incident are denoted by $O(i)$. Therefore, we have $O(i) = \{k : (i, k) \in A\}$. We assume a model of information transmissibility, in which a node i which is *exposed to* information can transmit it to one of its neighbors. Information transmission can take on many forms in practical settings:

(1) In a social network, information may be forwarded to any of the friends of a given user in the form of publicly visible text posts, hyperlinks, videos or messages. This user may or may not choose to adopt this piece of information and transmit it further.

(2) In a peer-to-peer recommendation or viral marketing system, a user may send a recommendation to any neighbor. The neighbor may or may not make a buying decision based on this recommendation. Furthermore, this recommendation may be forwarded to one of the neighbors of the node. In general, it has been observed [52] that customers in a network-marketing system have a certain value in terms of their being able to influence other members of the network. The determination of flow authorities will help us in identifying key points in the network which lead to the greatest spread of information.

(3) The above dynamic is generally true for a variety of network-based epidemic outbreaks, and may be generalized to social networks, blog posts [109], water monitoring systems, or any general network infection system which has structural similarity to epidemic outbreaks [34].

We will formally define the concept of *information exposure*.

Definition 4.1. *A node is said to be exposed to information bits $\mathcal{I}$, if at least one of its neighbors contains the information $\mathcal{I}$.*

It is important to note that the concept of neighborhood *information exposure* (as defined in this work) only entails the *presence of the information* at one of its neighbors, rather than any further explanation of what is done with it. The default assumption is that if a node contains some information bits, then all of its neighbors are automatically exposed to those bits. The probability that such an exposure results in *eventual* information assimilation is determined by a transmission matrix, which we will discuss shortly. We denote the transmission probability along edge (i, j) by p_{ij}. Note that this transmission probability simply indicates the probability that an exposure of node i also results in the *information being assimilated by* node j. Node j then automatically becomes eligible to transmit to its neighbors. We denote the corresponding matrix of transmission probabilities by $P = [p_{ij}]$. We note that this matrix is extremely sparse, because it is often overlaid on very sparse graphs such as social networks. We note that if r_i be the probability that a given node i contains information $\mathcal{I}$, then it *eventually* transmits the information $\mathcal{I}$ to adjacent node j with probability $r_i \cdot p_{ij}$. The value of p_{ij} can often be estimated from the underlying data.

In this work, we will examine the problem of picking a set of k points in the network which maximizes the aggregate probability of information assimilation over all nodes in the graph. We refer to these k nodes as the information authorities in the underlying network. We summarize the problem as follows:

Problem 4.1. *Determine the set S of k data points at which release of the information bits $\mathcal{I}$ would maximize the expected number of nodes over which $\mathcal{I}$ is assimilated.*

Algorithm *SteadyStateSpread*(Initial Set: S,

Transmission Matrix: P)

begin

for each $i \in S$ set $q^0(i) = 1$;

for each $i \notin S$ set $q^0(i) = 0$;

$t = 0$;

repeat

for each $i \in S$ set $q^{t+1}(i) = 1$;

for each $i \notin S$ **do**

begin

$q^{t+1}(i) = 1 - \prod_{l \in N(i)} (1 - p_{li} \cdot q^t(i))$;

end

$C_{t+1} = \sum_{i \notin S} |q^{t+1}(i) - q^t(i)|$;

$t = t + 1$;

until($C_t < 0.01 \cdot C_1$);

return($\sum_{i \notin S} q^t(i)$);

end

Figure 4.1: Determining the Expected Information Spread for a Given Starting Set of Nodes

We note that this is a particularly difficult problem, because the probability of the spread of the information at any particular node cannot be expressed easily in closed form. Rather, it is described in the form of a *non-linear system* of equations. We define $\pi(i)$ to be the steady-state probability that node i assimilates the information. Then, the expected steady state number of nodes which assimilate the information are given by $\sum_{i \in U} \pi(i)$. In order for node i to assimilate the information, it must receive the transmission from *at least* one of its neighbors.

The flip side of this argument is that in order for node i to not assimilate the information, it must not receive the transmission from *any* of its neighbors. The probability that none of the neighbors of node i transmit to it is given by $\prod_{l \in N(i)}(1 - \pi(l) \cdot p_{li})$. Therefore, we have:

$$1 - \pi(i) = \prod_{l \in N(i)} (1 - \pi(l) \cdot p_{li}) \tag{4.1}$$

In addition, for each of the k nodes in S at which the information is released, we set the corresponding value of $\pi(\cdot)$ to 1. Therefore, we have:

$$\pi(i) = 1 \quad \forall i \in S \tag{4.2}$$

The above system of equations is nonlinear, since it uses a product of the probability of (non-) exposure from different neighbors. This is a difficult set of equations to solve, and the corresponding result can only be obtained via numerical estimation. Furthermore, it is required to determine the set S optimally. The optimization problem is even more challenging. We will now restate Problem 4.1 more formally in terms of the relationships discussed above:

Definition 4.2. *Determine the set S of nodes which maximizes $\sum_{i \in U} \pi(i)$ subject to the following constraints:*

- $1 - \pi(i) = \prod_{l \in N(i)}(1 - \pi(l) \cdot p_{li}) \quad \forall i \notin S$
- $\pi(i) = 1 \quad \forall i \in S$

Next, we will describe a simple algorithm to determine the information authorities with the use of an iterative numerical method. Later, we will present a much faster probabilistic method for the same problem. This method uses a Bayes model in order to determine the optimal flow authorities.

4.4 Determining Optimal Information Flow Authorities

In this section, we will present algorithms for determining optimal information flow authorities. In order to determine optimal flow authorities, we also need to have a way to evaluate the (aggregate) steady state assimilation probability of all nodes, when the information is released at a *particular* set of nodes S. In order to design this algorithm, we will use an iterative algorithm in which $q^t(i)$ denotes the estimation of $\pi(i)$ in the tth iteration. This iterative approach is natural to solve the non-linear system of equations. In each iteration, we update the value of $q^t(i)$ from the value of $q^{t-1}(i)$ with the use of the equations in Definition 4.2. The overall algorithm is denoted by $SteadyStateSpread$ in Figure 4.1. The input to the algorithm is the set S at which the information is released.

The algorithm initializes $q^0(i) = 1$ for each node $i \in S$ and 0 for nodes which are not in S. Subsequently, an iterative approach is used to update the value of $q^{t+1}(\cdot)$ is updated from $q^t(\cdot)$ with the use of the equations in Definition 4.2. In each iteration, we track C_t, which is the aggregate change in the absolute probabilities from $q_t(\cdot)$ to $q_{t+1}(\cdot)$. The algorithm is terminated when the change in a given iteration is less than 1% of the change in the first iteration. At this point, it is assumed that the probability values have converged to values which are close to their true values.

The above method for determining the steady-state probabilities can be leveraged in order to determine the optimum set of k nodes at which the information should be released. We make use of a greedy approach which maximizes the expected increase in the information spread as calculated by Figure 4.1. The algorithm works with the use of an iterative approach

Algorithm RankedReplace(Transmission
Matrix: P, NumberOfAuthorities: k);
begin
Determine $SteadyStateSpread(\{i\}, P)$ for
each node i in the universal set U;
S =Initial set of k authority nodes with the
highest value of $SteadyStateSpread(\{i\}, P)$;
Sort nodes in $U - S$ in descending order of
$SteadyStateSpread(\cdot)$;
for each node i in $U - S$ in
descending order **do**
begin
Sort the list S in ascending order
of $SteadyStateSpread(\{j\}, P)$;
Pick the first element (if it exists) of
sorted list S which is such that
replacing i with it increases value of
$SteadyStateSpread(S, P)$
if no replacement has occurred in the last
r consecutive iterations, then
return(S) and terminate;
end
return(S);
end

Figure 4.2: The RankedReplace Algorithm

in which we start off with a candidate set of k nodes and continually increase its maximum flow value. We first pick the top k nodes with the largest individual steady state spread as the initial candidate set of flow authorities. Of course this way of picking the candidates ignores the structural relationships between these nodes. In general, we would like our flow authorities to be reasonably well separated from one another in order to maximize the probability of propagation of information throughout the social network. In order to achieve this goal, we use a *ranked replace* algorithm in which the nodes in $U - S$ are tried as possible replacements for nodes in S in decreasing order of their flow value.

The iterative portion of the algorithm proceeds as follows. We sort the nodes in $U - S$ in descending order of the steady state flow. In each iteration, we pick the next node i from $U - S$ and use it to replace a node in S, if such a replacement increases the total flow of S. Even though the flow value of i is typically less than that of the node it replaces, the total flow value may increase because of the nature of the network location of the two nodes. For this purpose, the nodes in S are tried as candidates for replacement in ascending value of $SteadyStateSpread(\cdot)$. The first replacement in this order which increases the objective function for the steady state information spread is executed. It is possible that no such replacement may exist. We continue to try different nodes in $U - S$ for replacement, until such attempts are unsuccessful for r consecutive iterations. At this point, the algorithm terminates, and the set of nodes S are reported as the flow authorities. The overall algorithm is illustrated in Figure 4.2. The algorithm is referred to as RankedReplace, which corresponds to the broad approach of ranking the nodes and iterative replacement based on the steady state flow impact.

4.4.1 The BayesTraceback Algorithm

The main problem with the solutions presented in the previous sections is that the algorithms require iterative determination of the steady-state probabilities. This can be rather slow in practice. In this subsection, we will discuss how to speed up the algorithms for determination of the information authorities. This algorithm provides an an approximation of the information authorities. The core-idea is to use a *random walk* based approach in which an information packet is viewed as a token, and it is assumed that the token at a given node j is inherited from one of its *incoming nodes* i with probability proportional to p_{ij}. Random walk modeling is used for the page rank problem, though this approach is different in the sense that we use it for *trace back* of the best *source of* information, rather than those nodes which will be visited often by a random surfer. Thus, the algorithm tends to be *backward looking from a desired result*, rather than *forward looking to determine the result*. In the experimental section, we will show that a direct application of the page rank model does not yield as accurate results as the *BayesTraceback* method.

The random walk model is a relaxation of the original model for two reasons:

(1) In the original model, a node can be infected only once, whereas a random walk can visit a node multiple times.

(2) In the original model, a given node may infect multiple nodes at once, whereas in this case, we are trying to trace the behavior of a single token, which is (stochastically) present only at one node at a time.

Algorithm BayesTraceback(Transmission Matrix: P

Discard Fraction: f, NumberOfAuthorities: k);

begin

$t = 0$;

for each node i set $q^0(i) = 1/n$;

repeat

$q^{-(t+1)}(j) = \sum_i q^{-t}(i) \cdot \frac{p_{ji}}{\sum_{l \in N(i)} p_{li}}$

$t = t + 1$;

Remove a fraction f of the nodes

from the graph with the least value of

$q^{-(t+1)}(\cdot)$, with the restriction

that at least k nodes should remain;

Scale up probabilities $q^{(-(t+1)}(\cdot)$ of

all remaining nodes by the same factor

so that the remaining probabilities sum to 1;

until(k nodes remain);

return remaining nodes;

end

Figure 4.3: The BayesTraceback Algorithm

We note that this simplification of the model allows us a trace-back of the steady-state probabilities with the use of a Bayes model. We will see that this approach is extremely efficient and provides a good approximation to the exact algorithm.

In the case of the random walk model, our aim is to pick k nodes in the data which are such that by releasing the information at these k points, the information spreads as evenly over the entire network as possible. Intuitively, this corresponds to release points which results in as

much of the network being disseminated with the information as possible. We note that the even spread of information may not be possible in steady-state, since the *steady-state* probabilities in a random-walk model are dependent upon the structure of the network and the transition probabilities, and are independent of the initial starting point probabilities. Nevertheless, our goal is to create an evenly spread probability distribution as an *intermediate transient* after a small number of iterations of the walk model. The goal is to find a set of k starting points which will create such an intermediate transient at some point. Therefore, for a network containing n nodes, we will start off with a *final transient probability distribution* of $1/n$ for each node, and then use the Bayes theorem repeatedly to *trace back* the initial probabilities for a certain number of iterations, and pick the k nodes with the largest apriori probability with the use of this traceback technique. Therefore, we start off with the probabilities for n nodes which are denoted by $\overline{q^0(\cdot)} = q^0(1) \ldots q^0(n)$. As noted earlier, each of these values is equal to $1/n$. In subsequent iterations, we will use the Bayes formula to determine the values of $\overline{q^{-1}(\cdot)}, \overline{q^{-2}(\cdot)} \ldots \overline{q^{-r}(\cdot)}$. Note that we use negative superscripts for the time component in order to denote the traceback starting from the 0th step of the walk. The vector $\overline{q^{-t}(\cdot)}$ indicates the probabilities after tracing the walk back for t steps.

Next, we will examine how the values of $\overline{q^{-(t+1)}(\cdot)}$ can be determined from $\overline{q^{-t}(\cdot)}$. For any particular node i, let us examine all the incoming edges from the corresponding node set $N(i)$. We note that the a-priori probability $P(j \rightarrow i| - t\text{th node} = i)$ that an information token at node i came from node j in the previous step of the random walk is given by the Bayes

formula over all possible nodes incoming into node i. Therefore, we have:

$$P(j \to i| - t\text{th node} = i) = \frac{p_{ji}}{\sum_{l \in N(i)} p_{li}} \tag{4.3}$$

In order to trace back the values of $\overline{q^{-(t+1)}(\cdot)}$ from $\overline{q^{-t}(\cdot)}$, we can examine the different cases over which the $-t$th node is i and sum up the values of $P(j \to i| - t\text{th node} = i)$ over these cases. Therefore, we have:

$$\begin{aligned} q^{(-(t+1)}(j) &= \sum_i q^{-t}(i) \cdot P(j \to i| - t\text{th node} = i) \\ &= \sum_i q^{-t}(i) \cdot \frac{p_{ji}}{\sum_{l \in N(i)} p_{li}} \end{aligned}$$

The second equation above simply traces back for the probability distribution of the position of the information token at time stamp $-(t+1)$ using the probability distribution of the token at time stamp $-t$. Therefore, we can start off with the evenly distributed probability vector $q^0(\cdot)$ and start tracing back the probabilities. The nature of the above probabilities suggest that nodes with high outdegree and outgoing probabilities will see increased probability during the traceback process. The process above is repeated for r iterations, and then the k nodes with the largest value of $q^{-r}(i)$ are picked as the correct candidates. It remains to describe the termination criterion. Furthermore, we need to design the algorithm in such a way, so that the algorithm converges.

It turns out that both of the above issues can be solved by making a heuristic change to the algorithm. This heuristic change speeds up the convergence and also provides a natural termination criterion to the algorithm. Note that since we only wish to determine the high probability nodes after the traceback, we can start removing the nodes, whose influence to this

is minimal. After each iteration of updating $q^{-(t+1)}(\cdot)$ from $q^{-t}(\cdot)$, we conceptually discard a fraction f of nodes with the least probability (least value of $q^{-(t+1)}(\cdot)$), by setting the corresponding values of $q^{-(t+1)}(\cdot)$ to zero. We also delete the corresponding nodes and edges from the graph. At the same time, we scale up the probabilities of the remaining nodes (by the same factor), so that they continue to sum to 1. This process is repeated iteratively until exactly k nodes are remaining. Note that the last iteration is special in the sense that less than a fraction f of the nodes may need to be dropped in order to ensure that we continue to have k nodes remaining. These k nodes are reported as the information authorities. The overall algorithm is illustrated in Figure 4.3. The input to the algorithm is the *discard fraction* f and the transition matrix P. The choice of the *discard fraction* f determines the speed of termination of the algorithm. A larger choice of f leads to faster convergence, but somewhat more inaccurate results. In practice, we chose f to be about 5% of the total number of nodes. We note that this algorithm is extremely efficient, since each iteration is a straightforward update step on the different nodes. Furthermore, for a graph containing n nodes, the maximum number of iterations is $\log(n/k)/\log(1/(1-f))$. This is because the number of nodes reduces by a factor of $(1-f)$ in each iteration, and the number of nodes need to be reduced from n to k in all iterations. Because of the logarithmic variation, this turns out to be quite modest. For example, for a network containing 10^6 nodes, $k = 10$ and $f = 0.05$, the total number of iterations is less than 180.

We note that successive removal of nodes and edges from the graph will eventually lead to the underlying graph becoming disconnected. This does not change the overall algo-

rithm, since the iterative transition relationships continue to hold within each connected component. Conceptually, the algorithm will eventually find the "most significant nodes" in the k highest probability components.

4.4.2 Restricting Source and Target Nodes

In the previous discussion, we picked the most relevant flow authorities for the entire set of nodes. In this section, we will examine the case when we wish to determine the flow authorities for a particular set T of target nodes. Such situations may arise in a number of scenarios in which a user may target a particular subset of nodes on which the information flow needs to be maximized. This problem can be achieved by simple modifications to each of the above algorithms:

(1) For the case of the RankedReplace algorithm, the only change is to modify the *SteadyState-Spread* algorithm. In the modified algorithm, we add the set T to the input parameters. The actual state probabilities on the nodes are computed using the same algorithm as before, except that the final information flow value which is returned is determined by summing up these probability only over T rather than the entire set of nodes. When the RankedReplace algorithm is executed with this new method of determining the steady state flow, it automatically picks the set of flow authorities which maximize the flow to the target set T.

(2) For the case of the BayesTraceback algorithm, we consider the nodes within target set as the sink nodes. Note that, the nodes that have the maximum influence over a set of target nodes intuitively correspond to the nodes that can evenly spread the information within the target nodes

as quickly as possible. However, to achieve the maximum flow within target nodes, we are free to take help of non-target nodes. Now, in this modified BayesTraceback approach, the algorithm still remains the same inside the subgraph imposed by the target nodes. For the subgraph imposed by the non-target nodes, we do not care the total flow that could be aggregated there by the whole process, as this is used only for the flow propagation within the target nodes. Therefore, the only change to the method is that we do not propagate the flow from target node to non-target node, but we propagate flow from non-target to target sets.

It is further possible to restrict the set of influential nodes to a particular set S. This situation can arise in cases, where the information can be released only at specific nodes. This generalization can be solved by adding this as an input parameter in case of RankedReplace algorithm. We only use the nodes in S for the ranking process in this case. For the case of the BayesTraceback algorithm, we run the algorithm in the same way as the previous case, except that the top-k nodes from the set S are picked as the final solution.

4.5 Experimental Results

We will present experimental results which illustrate the effectiveness, efficiency and robustness of our techniques on a number of real data sets. To compare our results, we consider some of the structural and random walk based algorithms as natural baselines. For example, we implemented the *Recursive Neighbor Mean (RNM) Algorithm* [120], which determines the peer influence groups and thereby identifies the dense clusters in a large network. The node with the highest degree centrality [51] in each cluster is considered the authority node in

Rank	RankedReplace	BayesTraceback	***Peer-Influence***	***Degree Discount IC***
1	W. Gao	W. Gao	L. Fortuna	W. Li
2	F. Catthor	P. S. Yu	D. Roy Chowdhury	W. Wang
3	P. S. Yu	M. T. Kandemir	Timothy D. Sullivan	L. Zhang
4	M. T. Kandemir	F. Catthoor	Wei Li	I. T. Foster
5	A. L. S. Vincentelli	A. L. S. Vincentelli	S. C. Lin	W. Zhang
6	E. Bertino	T. S. Huang	E. K. Zavadskas	M. Li
7	T. S. Huang	E. Bertino	K. J. Archer	L. Zhang
8	I. T. Foster	W.-Y. Ma	H. Van Keulen	L. Wang
9	L. Benini	D. F. Towsley	W. Wang	A. L. S. Vincentelli
10	H.-P. Seidel	I. T. Foster	R. Andrushkiw	J. Wang
11	W.-Y. Ma	E. D. Demaine	A. Thanachayanont	W. Liu
12	E. D. Demaine	H.-P. Seidel	H. Zimmermann	J. Zhang
13	M. Li	Ming Li	S. P. Perone	J. Wang
14	D. F. Towsley	V. Keulen	C. Lpez-Garca	L. Li
15	W. Wang	J. Han	M. McCormick	F. Catthoor
16	W. Li	H. Zhang	C. Jiang	Y. Zhang
17	M. Piattini	P. Nagley	L. F. Osborne	E. Bertino
18	H. Chen	J. Saltz	J. Dongarra	X. Li
19	L. Zhang	M. J. Irwin	J. P. Woodruff	W. Gao
20	H. Garcia-Molina	G. Weikum	A. Halme	H. Zhang

Table 4.1: Top-20 Results Obtained by Different Influence Maximization Methods

each cluster using this baseline approach. We refer to this algorithm as *Peer-Influence* in the experimental section. We also implemented the *Degree Discount IC* heuristic [39] discussed earlier. In the IC model, each active node gets a single chance to activate each of its neighbors independently with a certain probability. In the *Degree Discount* heuristic of the IC model, while selecting some node v as the authority node, we do not count the edge vu towards its degree, if u has already been selected as an authority node. Finally, we compare our top-k flow authority nodes with the top-k nodes having the highest *PageRank* [29] values.

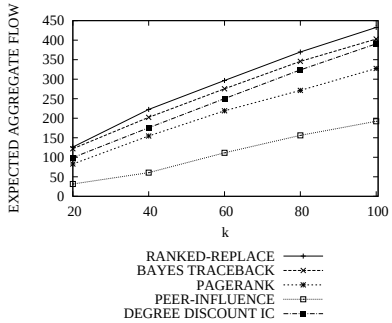


Figure 4.4: Influence Maximization: Effectiveness Results (DBLP)

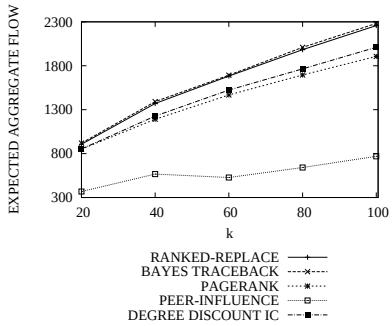


Figure 4.5: Influence Maximization: Effectiveness Results (Last.FM)

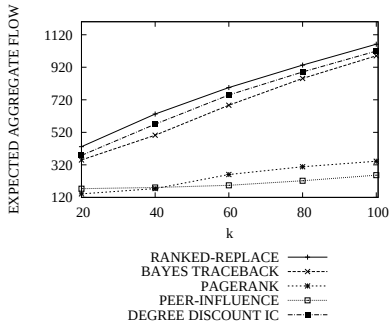


Figure 4.6: Influence Maximization: Effectiveness Results (Twitter)

4.5.1 Data Sets

The algorithms were tested on a variety of different kinds of interaction networks. These interaction networks were constructed from a number of different kinds of social net-

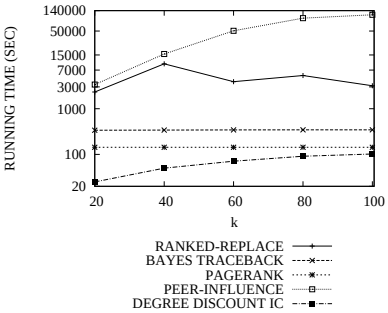


Figure 4.7: Influence Maximization: Efficiency Results (DBLP)

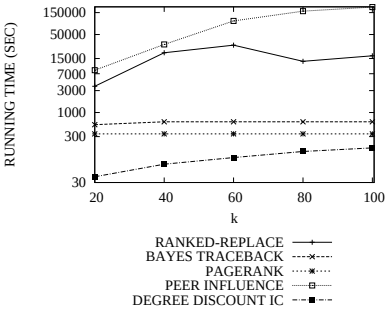


Figure 4.8: Influence Maximization: Efficiency Results (Last.FM)

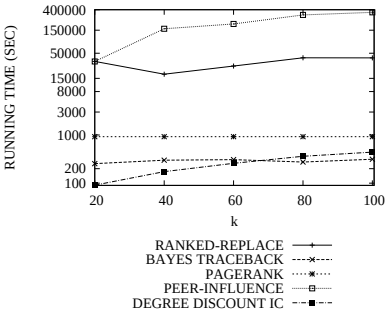


Figure 4.9: Influence Maximization: Efficiency Results (Twitter)

work settings. We describe the data sets in detail below.

DBLP Collaboration Network: We use the well known DBLP collaboration graph [47] consisting of $684,911$ distinct authors and $7,764,604$ collaboration edges among them. We define

the transmission probability of an edge to be proportional to the number of times that the two authors publish a paper together. The proportionality factor is the inverse of the maximum number of collaborations between any pair of authors in the network.

Last.fm Social Network: We crawled a social network consisting of $818,800$ users from the *last.fm* site. This is a music web site where users listen to their favorite tracks and communicate with each other based on their choice of music. There are a total of $3,340,954$ edges among these users. In each case, an edge represents user posts which correspond to song recommendations between users. The transmission probability of an edge is proportional to the number of times a recommendation was sent from one user to another. The proportionality factor is the inverse of the maximum number of communications between any two users.

Twitter Social Network: We crawled a social network consisting of $1,994,092$ users from http://twitter.com. Twitter is a free social networking and micro-blogging service that enables its users to send and read messages There are a total of $6,450,193$ edges among these users. In each case, an edge represents messages sent from one user to another. The transmission probability of an edge is proportional to the number of times the users have communicated. As in the previous case, the proportionality factor is the inverse of the maximum number of communications between any pair of users.

4.5.2 Case Studies

Before more concrete presentation of the effectiveness results with quantitative measures on the information spread, we will provide an intuitive exploration of the results obtained

with the different algorithms for the **DBLP** data set. This provides an intuitive understanding of the nature of the results obtained by the different methods. We provide the name of authority nodes for $k = 20$ in Table 4.1. It is evident that the authority nodes determined by the *Ranked-Replace* and BayesTraceback algorithms mostly contain well-known and influential researchers from different fields of computer science. Furthermore, we note that even though the algorithms are quite different from one another, the authority nodes determined are quite similar. Furthermore, these researchers are *structurally placed* in such a way so as to maximize the interaction with other researchers. All these factors contribute to the total aggregate flow across the entire network. The *Peer-Influence* method is particularly poor in determining good authority nodes, because it does not properly compute the flows on the basis of *random walk behavior*, and pure structural diameters simply do not encode enough information to ensure robustness. This results in lower aggregate flow across the whole graph. We also tested the *Degree Discount IC* algorithm. The *Degree Discount IC* algorithm determines better quality results than the *Peer-Influence* algorithm, because it uses a weighted version of random-walk, where the weight is determined by the degree of a node and the corresponding transmission probabilities; however, the determined authority nodes are quite different from the RankedReplace and BayesTraceback algorithms, because it performs a forward calculation as opposed to Bayes-based backward measures. This difference is quite significant; in the next section, we will use quantitative measures on the information spread to show that the RankedReplace and BayesTraceback algorithms are more effective than the *Degree Discount IC* algorithm in many cases.

4.5.3 Effectiveness Results

In order to measure the effectiveness of a set of authority nodes S, we used expected value of the steady state flow from the determined set S to the remaining set of nodes. We determine the expected aggregate flow for different values of k, the number of authority nodes. Figure 4.4 illustrates the effectiveness result for the **DBLP** data set. The value of k is illustrated on the X-axis, whereas the flow value is illustrated on the Y-axis. The expected aggregate flow increases with the number of authority nodes, since the release of information at a larger number of nodes leads to greater spread of information. The *Ranked-Replace* method slightly outperforms the *BayesTraceback* algorithm. We will see that the *BayesTraceback* method is also extremely efficient, and therefore it is the most practical alternative among the different methods. Furthermore, both techniques perform *significantly* better than the three baseline techniques. For example, when we set $k = 60$, the expected aggregate flow using the *Ranked-Replace*, *BayesTraceback*, *PageRank*, *Degree Discount IC* and the *Peer-Influence* methods are 296.70, 275.42, 211.67, 250.28 and 111.48 respectively.

Figure 4.5 illustrates the effectiveness results of our method for the **last.fm** data set. Both the *Ranked-Replace* and BayesTraceback algorithms perform very similarly, and also significantly outperform the three baseline methods. For $k = 60$, the expected aggregate flow using the *Ranked-Replace*, BayesTraceback, *PageRank*, *Degree Discount IC* and the *Peer-Influence* methods are 1682.62, 1692.21, 1450.62, 1523.50 and 527.27 respectively. In Figure 4.6, we illustrate the results for the **Twitter** data set. In this case, the *Degree Discount IC* heuristic performs slightly better than the BayesTraceback method. For example, for $k = 80$, the expected

aggregate flow using the RankedReplace, BayesTraceback, *PageRank*, *Degree Discount IC* and the *Peer-Influence* methods are 933.64, 851.47, 258.76, 891.24 and 222.34 respectively.

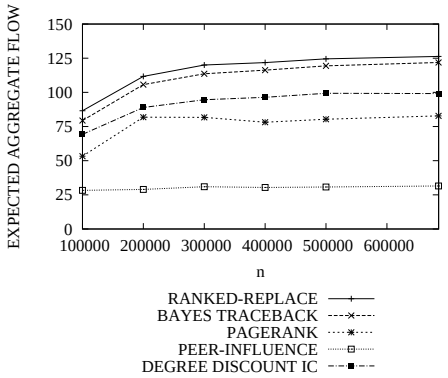


Figure 4.10: Influence Maximization: Effectiveness vs. Network Size (DBLP)

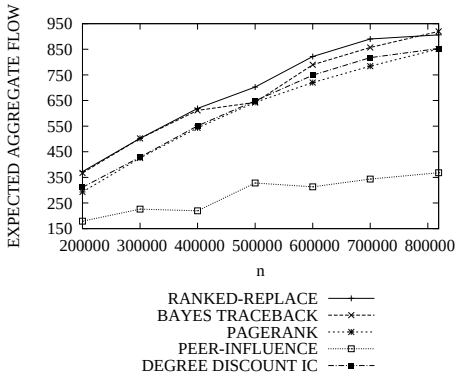


Figure 4.11: Influence Maximization: Effectiveness vs. Network Size (Last.FM)

4.5.4 Efficiency Results

We compare the running time of the RankedReplace and *BayesTraceback* methods with that of the three baseline methods. Figure 4.7 shows the efficiency result for the **DBLP** data set. The number of authority nodes k is varied from 20 to 100 on the X-axis, whereas the

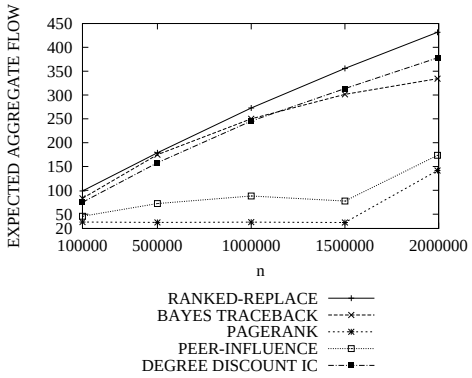


Figure 4.12: Influence Maximization: Effectiveness vs. Network Size (Twitter)

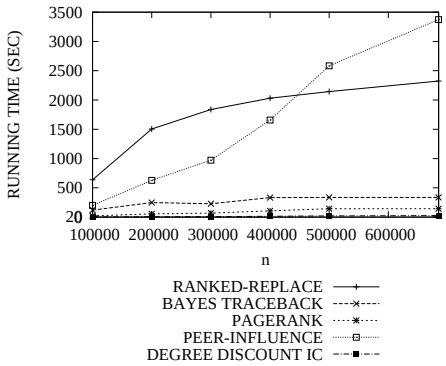


Figure 4.13: Influence Maximization: Efficiency vs. Network Size (DBLP)

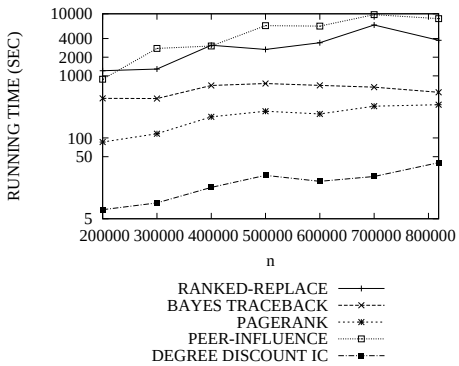


Figure 4.14: Influence Maximization: Efficiency vs. Network Size (Last.FM)

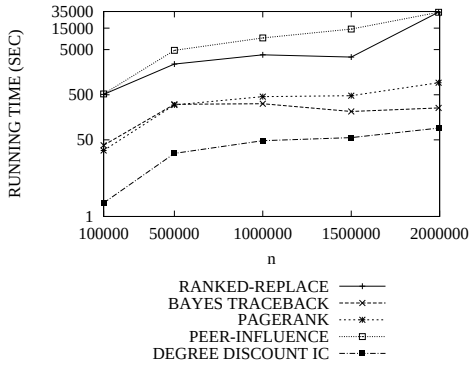


Figure 4.15: Influence Maximization: Efficiency vs. Network Size (Twitter)

running time is illustrated on the Y-axis. The *Peer-Influence* approach is the most inefficient, and its running time increases rapidly with the number of authority nodes. The BayesTraceback algorithm, on the other hand, is very efficient, though the *Degree Discount IC* algorithm is the fastest. For $k = 60$, the running time of the RankedReplace, *BayesTraceback*, *PageRank*, *Degree Discount IC* and *Peer-Influence* methods are 3904, 343, 104, 71 and 50743 seconds respectively. The running times for the **last.fm** data set are illustrated in Figure 4.8. For $k = 60$, the running time of the *Ranked-Replace*, BayesTraceback, *PageRank*, *Degree Discount IC* and the *Peer-Influence* method are 29265, 628, 301, 105 and 98930 seconds respectively.Thus, the *Peer-Influence* method is two orders of magnitude slower than our two methods. Also, the BayesTraceback method is an effective alternative to the RankedReplace method, while maintaining a significantly high level of effectiveness.

Besides, the time requirement for the BayesTraceback method does not vary much with respect to k. This is because a fixed fraction of the nodes are discarded in each iteration, and the number of iterations for convergence of this method is inversely proportional to $\log n$.

The *Ranked-Replace* method is a greedy approach and it does not follow any specific pattern with respect to k. However, the running time of the *Degree Discount IC* is proportional to k [39]. We note that, in the **Twitter** dataset (Figure 4.9), the BayesTraceback approach is more efficient than the *Degree Discount IC* heuristic for higher values of k. For example, when we set $k = 80$, the running times of the *Ranked-Replace*, *BayesTraceback*, *PageRank*, *Degree Discount IC* and the *Peer-Influence* methods are 40225, 275, 1001, 362 and 312345 seconds respectively.

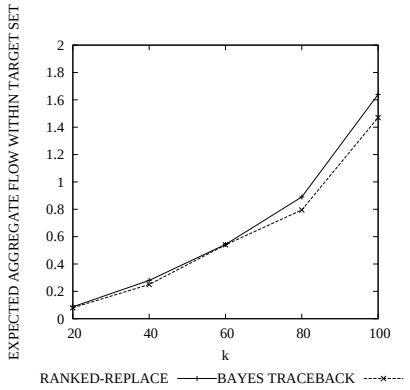


Figure 4.16: Maximum Aggregate Flow for Particular Target Nodes (DBLP)

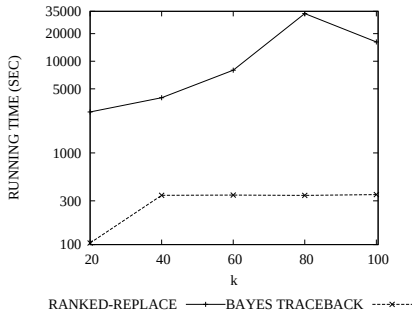


Figure 4.17: Running Time to Find Authority Set for Particular Target Nodes (DBLP)

4.5.5 Robustness and Scalability with increasing Network Size

Our goal in this section is to test the robustness and scalability of the method with increasing network size. This will show the effectiveness of the method with different network sizes, and also the scalability of the method. In order to obtain networks of increasing sizes, we randomly deleted nodes (and their incident edges from the data set), and tested the algorithm over networks of increasing size. We set k, the number of authority nodes, as 20. We provide results on the effectiveness and efficiency for increasing number of nodes.

Figure 4.10 shows the variation of the maximum information spread with increasing number of nodes for the **DBLP** data set. The total flow value initially increases with the number of nodes, because the full benefit of multiple points of information release in small networks is not realized. On the other hand, if the networks are too large, then the information spread may get sufficiently damped in a few iterations. Therefore, the flow value increases relatively fast up to the value of $n = 300,000$ for both the *Ranked-Replace* and BayesTraceback methods, and then levels off. The *Ranked-Replace* and BayesTraceback method both outperform the baseline approaches by a high margin for all values of n. This suggests that the method is extremely robust over networks of different sizes. Figure 4.11 shows the corresponding results for **last.fm** data set. As in the case of the **DBLP** data set, the expected aggregate flow for the *Ranked-Replace* and BayesTraceback methods are much higher than that of *Degree Discount IC, PageRank* and the *Peer-Influence* methods over the entire range of possible network sizes. We plot the expected aggregate flow for different network sizes for the case of the **Twitter** data set in Figure 4.12. The results are similar to the case of the other two data sets when the number

of nodes is less than $1,000,000$. However, for n greater than $1,000,000$, the *Degree Discount IC* heuristic performs slightly better than the BayesTraceback method.

Figure 4.13 illustrates the running time scalability with increasing number of nodes for the **DBLP** data set. For the BayesTraceback method, the running time does not vary much with respect to the number of nodes since the number of iterations increases only logarithmically with the number of nodes. As observed earlier, it is slower than the *Degree Discount IC* method but significantly faster than either the RankedReplace or the *Peer-Influence* methods. For small values of the number of nodes n, the *Peer-Influence* approach is slightly faster than the *Ranked-Replace* method; however, the former does not scale well and is much slower than the *Ranked-Replace* method for larger networks. For $n = 500,000$ or higher, the *Peer-Influence* approach requires significantly more time than the *Ranked-Replace* technique. The results for the **last.fm** data set show similar trends, as is evident from Figure 4.14. The results for the **Twitter** data set are illustrated in Figure 4.15. In this case, the running times of *Degree Discount IC* and *BayesTraceback* algorithms are comparable. These algorithms are also significantly faster than the other two methods.

4.5.6 Targeted Flow Authorities

We also tested our two schemes for the case when we determined the flow authorities for a particular set of target nodes. For the **DBLP** collaboration graph, we randomly selected a set of 1000 target nodes and determine the corresponding authority nodes which will maximize the flow within that target set. Figure 4.16 illustrates the expected aggregate information spread

within this target set (for different number k of authority nodes) using the modified *Ranked-Replace* and BayesTraceback methods. Figure 4.17 illustrates the corresponding running time to find the authority nodes. In this case, we do not show the baseline methods because they cannot be easily modified when particular nodes are targeted. Thus, our scheme also provides better functionality than the baseline methods. It is evident that the modified *BayesTraceback* method performs almost as well as the modified *Ranked-Replace* technique; however it is significantly faster in terms of running time. Unlike the RankedReplace method, the running time of the BayesTraceback method is relatively insensitive to the number of authority nodes k. Therefore, the BayesTraceback method provides the best tradeoffs between quality and efficiency.

4.6 Summary

In this chapter, we designed an algorithm for the determination of optimal flow authorities in social networks. We designed two algorithms for the task, which correspond to the RankedReplace and BayesTraceback algorithms. We presented experimental results illustrating the effectiveness of our methods on a number of social networking and collaboration graphs. Our results show that the techniques proposed in this work are much more effective than the currently available techniques. While the RankedReplace technique is slightly more effective than the BayesTraceback method, the latter is significantly more efficient. Furthermore, it is much superior to the baseline methods in terms of effectiveness. The BayesTraceback algorithm provides the best tradeoff between quality and efficiency.

Chapter 5

Graph Pattern Mining

> *"Nature uses only the longest threads to weave her patterns, so that each small piece of her fabric reveals the organization of the entire tapestry."*
>
> Richard P. Feynman

Mining graph patterns in large information networks is critical to a variety of applications such as malware detection and biological module discovery. However, frequent subgraphs are often ineffective to capture association existing in these applications, due to the complexity of isomorphism testing and the inelastic pattern definition.

In this chapter, we introduce proximity pattern which is a significant departure from the traditional concept of frequent subgraphs. Defined as a set of labels that co-occur in neighborhoods, proximity pattern blurs the boundary between itemset and structure. It relaxes the rigid structure constraint of frequent subgraphs, while introducing connectivity to frequent itemsets. Therefore, it can benefit from both: efficient mining in itemsets and structure proximity from graphs. We developed two models to define proximity patterns. The second one, called

Normalized Probabilistic Association (NmPA), is able to transform a complex graph mining problem to a simplified probabilistic itemset mining problem, which can be solved efficiently by a modified FP-tree algorithm, called pFP. NmPA and pFP are evaluated on real-life social and intrusion networks. Empirical results show that it not only finds interesting patterns that are ignored by the existing approaches, but also achieves high performance for finding proximity patterns in large-scale graphs.

5.1 Introduction

Graph patterns are building blocks for several key graph applications, including graph indexing, graph search, graph classification and clustering [40, 50, 160, 167]. Existing graph pattern mining algorithms, like those developed in [28, 79, 81, 84, 101, 124, 150], achieved great success using strategies that efficiently traverse the pattern space. However, the definition of frequent subgraphs might not be appropriate for new application scenarios present in social and information networks. First, the definition is not elastic enough to capture fuzzy patterns existing in massive attributed graphs. Figure 5.1 shows one example, where each node is attached with a set of labels. These labels can be movies recommended by a user, functions carried by a gene, or intrusions initiated by a computer. As illustrated in Figure 5.1, a, b, c often occur together and formulate an association pattern, while d, c are not associated together. However, $\{a, b, c\}$ is neither a frequent subgraph, nor a frequent itemset if we treat each node as a transaction. Pattern $\{a, b, c\}$ has three characteristics: (1) Proximity, these three labels are tightly connected; (2) Frequency, they appear many times; (3) Flexibility, they are not always con-

nected in the same way. Due to these characteristics, we can not apply the traditional frequent graph mining algorithms such as FSG [101] and gSpan [159] to find them. On the other hand, frequent itemset mining [12, 73] can not be used either, since $\{a, b, c\}$ do not appear in the same set of nodes.

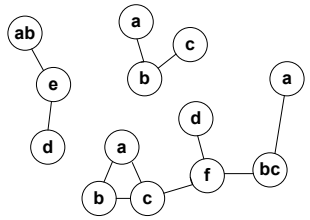


Figure 5.1: Proximity Pattern $\{a, b, c\}$

Secondly, for small graphs such as chemical structures, isomorphism checking is never a problem as demonstrated by the existing frequent graph mining algorithms. However, for large graphs like intrusion networks and social networks, there can be a huge set of isomorphic embeddings existing for frequent subgraphs. It becomes costly to generate all kinds of frequent subgraphs. To overcome the above two issues, we propose a new graph pattern concept, called *Proximity Pattern*. A proximity pattern is a subset of labels that repeatedly appear in multiple tightly connected subgraphs in G. $\{a, b, c\}$ in Figure 5.1 is an example. Proximity pattern is an itemset. However, it has a connectivity requirement: the labels must be associated tightly and frequently in the graph. For example, in a social network, it can be a set of movies that are watched by multiple groups of users. That is, in order to find proximity patterns among movies, one should not only consider the collection of movies watched by each person (in this case, it is a traditional itemset mining problem); instead, one should also consider the movies watched by his or her friends and friends of friends. In this case, labels associated with two

different nodes are related due to the connection between these two nodes. The same mining problem also exists in finding associations of intrusions on the Internet, where each node corresponds to an IP address and there is a directed edge between two IP addresses if an intrusion attack takes place between them. It is interesting to find the association of different attack types, which can be used to analyze intrusions.

In this work, we first introduce an intuitive neighbor association model to define and allocate proximity patterns by identifying the embeddings of these patterns in a graph and then finding a weighted maximum independent set among these embeddings. Although this approach is intuitive, it is inefficient to find patterns in large graphs due to the complexity of embedding enumeration and maximum independent set finding. Therefore, we redefine proximity patterns from an influence point of view, using a probabilistic information propagation model. Based on this model, we propose novel techniques for finding proximity pattern within a large graph, which consider conditional probabilistic association of the labels at each vertex. In the end, a statistical test is developed to measure the significance of discovered proximity patterns.

Our Contributions. To the best of our knowledge, this is the first research work introducing the concept of proximity patterns in large graphs.

We model the problem of determining the proximity among labels in two distinct approaches, neighbor association and information propagation. While the neighbor association model is a direct approach of finding the association among labels based on their distance across the edges of the graph, we have shown that this method is not efficient for large scale graphs. In the information propagation model, we develop novel probabilistic techniques to

determine the proximity among labels in a graph database, based on the Markov model [128]. We justify that they will be efficient as well as consistent under interpretations of "relation between transactions" and the "association of labels". The propagation model is able to transform a complex graph mining problem to a simplified probabilistic itemset mining problem, which can be solved efficiently by a modified FP-tree algorithm, called pFP(probabilistic FP-growth). Furthermore, for the discovered patterns, we define an objective function that will measure their interestingness using randomized test.

In summary, we propose a complete pipeline to define and mine proximity patterns in massive graphs in a scalable manner. As tested in real-life social networks and intrusion networks, proximity patterns turn to be interesting and are able to capture patterns missed by frequent itemsets and frequent subgraphs.

5.2 Related Work

Finding graph patterns is an active research topic in data mining. In the area of mining a set of graphs, efficient frequent subgraph mining algorithms have been proposed, including AGM [84],FSG [101], gSpan [159], followed by Path-Join, MoFa, FFSM, GASTON, etc. Recently, techniques were developed to mine maximal graph patterns [82] and significant graph patterns [75]. These methods adopt subgraph isomorphism testing as a way to count the support of graph patterns in multiple graphs.

In the area of mining single massive graphs, [35, 60, 102] developed techniques to calculate the support of graph patterns, i.e., how many times we should count a subgraph in

one graph, when there are overlapping embeddings. Kuramochi and Karypis [102] proposed using the maximum independent set as the support of subgraphs, which is proved to have the downward closure property by [60]. [19] proposed a support measure that is computationally less expensive and often closer to intuition than other measures. Since subgraph isomorphism is still used in these methods, they cannot handle the proximity patterns discussed in this work, where strict isomorphism is not desired.

Discovering rules from transactions has been extensively studied. The concept of association rules was first introduced in [11, 13], where the authors proposed an Apriori based approach to determine all frequent itemsets. [129] describes a hash-based algorithm which is an improvement over the Apriori approach. In [164], Zaki proposed a depth-first search algorithm using set intersection. FP-growth was introduced by Han et al. in [73], which uses an extended prefix-tree (FP-tree) structure to store the database in a compressed form. In [18], Au and Chan introduced fuzzy association rules based on the fuzzy set theory. Here, each item is assigned a non-binary weight according to its significance with respect to a user defined criterion. In [116], Mangalampalli and Pudi have shown how the existing algorithms like Apriori and FP-growth can be modified to mine data in a fuzzy environment. Very recently, Bernecker et al. [24] and Charu et al. [8] proposed techniques for mining frequent itemsets from uncertain databases. Their techniques could also be applied. However, to the best of our knowledge, no previous work targets the problem of finding proximity patterns in the context of massive graphs.

5.3 Preliminaries

An attributed graph $G = (V, E)$ has a label set L and each node is attached with a set of labels. The label set of a node u in G is $L(u)$. Let I be a subset of labels such that the labels in I tightly connect and appear repeatedly in G. I is named as *"Proximity Pattern"*. Proximity patterns are degenerated to frequent itemsets, if we drop all the edges in G. In this work, we focus on *bidirectional* and *unweighted* graphs. However, the proposed models and algorithms can be applied to *directed* graphs as well. Some modifications are required for *weighted* graphs, which we shall discuss later in Section 5.5.3.

Let $D = \{t_1, t_2, \ldots, t_m\}$ be a set of *independent transactions* (in the context of attributed graphs, the set of nodes). Each transaction contains a subset of items in L.

Definition 5.1 (Support)**.** *The support $sup(I)$ of an itemset $I \subseteq L$ is the number of transactions in the data set that contain I. Sometimes, we also use the percentage to represent support.*

An itemset is called *frequent* if its support is greater than a user-defined minimum threshold. Nearly all the classical frequent itemset mining algorithms apply the property of *Downward Closure* [12] to prune the pattern search space.

Definition 5.2 (Downward Closure)**.** *For a frequent itemset, all of its subsets are frequent; and thus for an infrequent itemset, all of its supersets must be infrequent.*

Unfortunately, since frequent itemset mining does not consider the connections in an attributed graph, it might miss interesting patterns. Figure 5.2 shows an example. If we consider

each node as an independent transaction, $\{l_1, l_2\}$ will not be reported as a frequent itemset. The two items do not occur together in any of the nodes. However, a careful examination of Figure 5.2 reveals that they always occur within one-hop distance of each other. $\{l_1, l_2\}$ is a proximity pattern: l_1 is associated in the proximity of l_2.

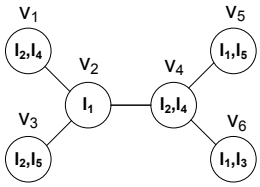


Figure 5.2: Frequent Itemset vs Proximity Pattern

For a proximity pattern I, we need to identify locations of this pattern in G. Each of these locations shall contain all of labels in I.

Definition 5.3 (Embedding and Mapping)**.** *Given a graph G and a subset of vertices π, $\pi \in V(G)$, Let $L(\pi)$ be the set of labels in π, i.e., $L(\pi) = \cup_{u \in \pi} L(u)$. Given a label subset I, π is called an embedding of I if $I \subseteq L(\pi)$. A mapping ϕ between I and the vertices in π is a function $\phi : I \rightarrow \pi$ s.t., $\exists\, l,\ \phi(l) \in \pi$ and $l \in L(\phi(l))$. A mapping is minimum if it is surjective, i.e., $\forall v \in \pi, \exists l$ s.t. $\phi(l) = v$.*

In Figure 5.2, $\{v_1, v_2, v_3\}$ forms an embedding of $\{l_1, l_2, l_5\}$. There can be two possible mappings in this embedding: (1) ϕ_1 maps l_1 to v_2, l_2 to v_1, and l_5 to v_3, and (2) ϕ_2 maps l_1 to v_2, l_2 to v_3, and l_5 to v_3. In these two mappings, ϕ_1 is minimum, ϕ_2 is not. The vertices in π might not be connected. For example, $\{v1, v3\}$ is an embedding of $\{l_4, l_5\}$

Given an itemset I and a mapping ϕ, we need a function $f(\phi)$ to measure its association strength: how tightly the mapped labels in π are connected. For example, $f(\phi)$ could be the inverse of diameter of ϕ or the inverse of $\sum_{u,v \in V(\phi)} d(u,v)$, where $d(u,v)$ is the shortest distance between u and v. Since there could be multiple mappings in π, we always choose the mapping that has the highest value of $f(\phi)$. To simplify the presentation, we also denote the strength of an embedding as $f(\pi)$.

In the next section, we are going to investigate two models to define the support of proximity patterns.

5.4 Neighbor Association Model

The complexity of proximity patterns rises from the interconnections of labels in a graph. One has to perform the following three steps to identify proximity patterns:

1. Find all the embeddings, $\pi_1, \pi_2, \ldots, \pi_m$ of an itemset I in the graph,
2. For each embedding π, measure its strength $f(\pi)$,
3. Aggregate the strength of the embeddings, $F(I) = \sum_{i=1}^{m} f(\pi_i)$. Take $F(I)$ as the support of I.

In order to find the support of a proximity pattern, one has to first enumerate all the embeddings of the pattern. Unfortunately, due to graph connections, there could be an exponential number of redundant embeddings. First, the boundary between the embeddings of a pattern is not obvious. When two embeddings overlap, the overlapped part might be double

counted. The support derived from multiple embeddings will violate the downward closure property (Definition 5.2). That is, the support of a pattern I might be less than a pattern I', even though $I \subseteq I'$, which makes it difficult to design fast mining algorithms. Secondly, any subset of vertices, π, could be an embedding of a pattern I as long as $I \subset L(\pi)$, though for those loosely connected embeddings, their strength might be negligible.

In order to solve the above two issues, we introduce two models in this chapter, neighbor association model and information propagation model.

Let $\pi_1, \pi_2, \ldots, \pi_m$ be the embeddings of I in G. we build an overlapping graph: each node represents an embedding and an edge connects two embeddings if they share at least one common vertex. In the overlapping graph, each node has $f(\pi)$ as its weight. Figure 5.3 shows an example of a partial overlapping graph derived from Figure 5.2.

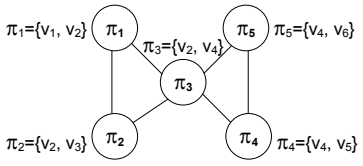


Figure 5.3: Overlapping Graph

For frequent graph mining in a single graph, Kuramochi and Karypis [102] proposed using the maximum independent set as the support of subgraphs, which is proved to have the downward closure property [60]. An independent set in a graph is a subset of vertices with no edge connecting them. In Figure 5.3, embeddings π_1, π_4 form a maximum independent set. This concept can be extended to the overlapping graph with weights. For a label set I, the support of I could be the sum of vertex weights derived by the maximum weight independent

set. It can be proved that the support defined using maximum weight independent set has the downward closure property too. We call this model *Neighbor Association Model*.

While the neighbor association model solves the pattern overlapping issue, it is NP-hard in general with respect to the number of embeddings for a given pattern [161]. Since the number could be huge, in practice, it is not feasible to generate all the embeddings of proximity patterns and then find their maximum weight independent set. Thus we resort to the second model, *Information Propagation Model*.

5.5 Information Propagation Model

The neighbor association model examines the association from a graph structure perspective. For example, for two labels l_1, l_2, in a graph, how closely they are connected and how often they are connected. It is possible to examine the same problem from a network influence perspective. Take a movie recommendation social network as an example, where users could recommend movies to their friends. Assume G_0 is the initial graph. Based on the recommendations, users might watch more movies and generate a new graph G_1 with updated watched movie lists. This process iterates until it reaches a stable graph where the movie list for each user does not change any more.

$$G_0 \rightarrow G_1 \rightarrow \ldots \rightarrow G_n.$$

In an ideal situation, it is meaningful to mine frequent itemsets in G_n. However, in reality we only have an incomplete snapshot between G_0 and G_n. Proximity patterns in G_i

could be interpreted as an approximation to frequent itemsets in G_n. With that being said, if we are able to simulate the influence process by generating $\widetilde{G}$ from G_i to approximate G_n, we can instead use frequent itemsets mined from $\widetilde{G}$ to represent proximity patterns in G_i. This is the main idea of the information propagation model.

We model the influence process using a first order Markov model. The given graph is considered as the present state, and the association among labels in the future state will be reached through an iterative stochastic process. Let $L(u)$ be the present state of u, denoted by the labels present in u, and l be a distinct label propagated by one of its neighbors and $l \notin L(u)$. Hence, the probability of observing $L(u)$ and l is written as

$$P(L \cup \{l\}) = P(L|l)P(l), \tag{5.1}$$

where $P(l)$ is the probability of l in u's neighbors and $P(L|l)$ is the probability that l is successfully propagated to u.

For multiple labels, $l_1, l_2, \ldots, l_m$, the joint probability of observing $L \cup \{l_1, \ldots, l_m\}$ can be written as, assuming each label is propagated independently,

$$P(L \cup \{l_1, \ldots, l_m\}) = P(L|l_1) * \ldots * P(L|l_m) * P(l_1) * \ldots P(l_m). \tag{5.2}$$

The propagation model captures an important characteristic in social graphs where nodes can influence each other. As the distance increases, the influence decreases [4], which is exactly what proximity patterns would like to capture. In the next two subsections, we introduce two distinct approaches to assign values to the aforementioned conditional probabilities,

$P(L|l)$, along with the detailed algorithms. These two approaches handle the situation when the same label is propagated by multiple nodes, with different distances.

5.5.1 Nearest Probabilistic Association

According to the exponential decay model of transmissibility [156], the transmissibility decays as a power of the distance from the initial source. In the *Nearest Probabilistic Association* model (NPA), the conditional probability $P(L(u)|l)$ of Eq. 5.1, $A_u(l)$, is defined as follows.

Definition 5.4 (Nearest Association)**.** *Let l be a label present in v which is the nearest one to u, where $l \notin L(u)$. $A_u(l) = P(L(u)|l) = e^{-\alpha \cdot d}$, where d is the distance from v to u, and α is the decay constant ($\alpha > 0$).*

$A_u(l)$ decays to zero as d approaches to ∞. For an unweighted graph, we assume $d = 1$ for each edge. The algorithm to find the stable propagated graph $\widetilde{G}$ is outlined in Algorithm 3. $\widetilde{G}$ is like a classical transaction database, where each node represents a transaction and each label represents an item. However, unlike transactions, in $\widetilde{G}$, items could have values 0, 1 or a fraction between them due to the probabilistic property of our model. Similar to classical transactions, 1 denotes full association and 0 no association; whereas a proper fraction indicates partial association of the labels at that vertex. The association value should decrease as the distance between a vertex and a label increases [59, 106]. Therefore, we have an input cut-off parameter ϵ in Algorithm 3. We do not propagate a label when the nearest association value for that label is less than ϵ.

Algorithm 3 Generate Intermediate Dataset $\widetilde{G}$

Input: Graph G, cut-off parameter ϵ.

Output: Intermediate Dataset $\widetilde{G}$.

1: $i = 0$ // iteration

2: **for all** vertex u of G **do**

3: Let $L_0(u)$ be the label set of u

4: $\forall l \in L_0(u)$, $A_u(l)$=1; otherwise $A_u(l)$=0

5: **end for**

6: **for all** vertex u of G **do**

7: **for all** label l in $L_i(v) \setminus L_i(u)$, v is u's neighbor **do**

8: update $A_u(l)$ using Definition 5.4 (choose the maximum one)

9: If less than ϵ, do not propagate l to u

10: **end for**

11: $L_{i+1}(u) = \{L_i(u) \cup \{l\} | A_u(l) > 0\}$

12: **end for**

13: **if** $L_{i+1} = L_i$ for all vertices in G **then**

14: Output A_u for all $u \in V(G)$

15: **else**

16: $i = i + 1$, goto step 2

17: **end if**

Note that $A_u(l) = 1$ when u itself has the label l; $A_u(l) = 0$ when l is considerably away from u, or there is no path from u to any of the vertices having l. Since the association of a label at a vertex is determined by the nearest occurrence of the label, we call it "nearest association". Once the intermediate dataset is formed, following the joint distribution of Eq. 5.2, we shall define the support of a proximity pattern.

Definition 5.5 (Probabilistic Support)**.** *Given an intermediate dataset $\widetilde{G}$ derived by the Nearest Probabilistic Association model, the support of $I = \{l_1, l_2, \ldots, l_m\}$, $sup(I) = \frac{1}{|V|} \sum_{u \in V} A_u(l_1) \cdots A_u(l_m)$, where $A_u(l)$ represents the probability of observing l at u.*

The support definition in NPA has the downward closure property. That is, $sup(I) \geq sup(J)$ if $I \subseteq J$. This is due to the fact that $A_u(l) \leq 1$. Let $I = \{l_1, l_2, \ldots, l_m\}$ and $J = \{l_1, l_2, \ldots, l_m, l_{m+1} \ldots, l_n\}$. Since

$$\prod_{i=1}^{m} A_u(l_i) \geq \prod_{i=1}^{m} A_u(l_i) \prod_{i=m+1}^{n} A_u(l_i),$$

we have

$$sup(I) \geq sup(J).$$

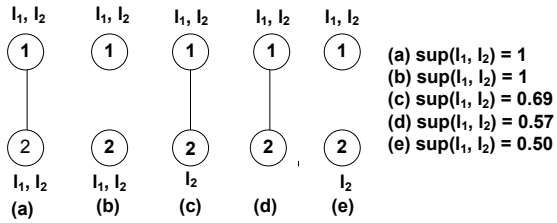


Figure 5.4: Consistency: NPA and Frequent Itemset

The definition is also consistent with the support definition of frequent itemsets, where $A_u(l)$ can only be 0 or 1. Figure 5.4 shows the connection, where the decay constant α is set at 1. NPA rightly assigns the highest support value $(= 1)$ for $\{l_1, l_2\}$ in Figure 5.4(a) and 5.4(b), which is consistent with frequent itemsets. The support value gradually decreases in Figure 5.4(c), 5.4(d), and 5.4(e). The decreasing order of support reflects the association strength of l_1, l_2 in different structures. Figure 5.4(c) has a higher support for $\{l_1, l_2\}$ than Figure 5.4(d) since there are two l_2's close to l_1. Note that, we assume $\alpha = 1$ for all these examples.

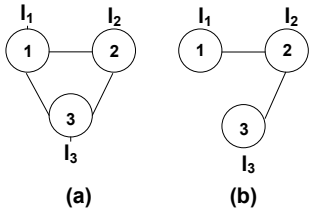


Figure 5.5: Support vs Structure Difference

The NPA support is both commutative and associative. It can also tell slight difference between structures. Table 5.1(a) and 5.1(b) show the intermediate dataset for two different substructures in Figure 5.5(a) and Figure 5.5(b) respectively. Rightly this approach assigns higher support for $\{l_1, l_2, l_3\}$ in Figure 5.5(a).

Table 5.1(a)			
	l_1	l_2	l_3
$node\_1$	1	0.37	0.37
$node\_2$	0.37	1	0.37
$node\_3$	0.37	0.37	1
$sup(l_1, l_2, l_3) = 0.14$			

Table 5.1(b)			
	l_1	l_2	l_3
$node\_1$	1	0.37	0.14
$node\_2$	0.37	1	0.37
$node\_3$	0.14	0.37	1
$sup(l_1, l_2, l_3) = 0.08$			

Table 5.1: NPA Intermediate Dataset for Figure 5.5

Complexity. Let $|V|$ be the total number of vertices in G, the average degree of each vertex be d, and the average number of labels in each vertex be s. If there are total t iterations in Algorithm 3, the time complexity of generating the intermediate dataset $\widetilde{G}$ is $O(|V| \cdot d^t \cdot s)$. Since $t << |V|$, the complexity is almost linear in the number of vertices. The parameter t is a measure of the maximum depth where we may look for a label. The depth will be determined by the decay constant α and ϵ. In social networks, the mutual interaction and social influence usually decays quickly with distance t [4, 31, 78, 118]. The influence is negligible when $t > 3$. In NPA, the probabilistic association value of a label at a distance t is given by $e^{-\alpha \cdot t}$. Since we ignore the value less than ϵ, $t \leq \frac{1}{\alpha} \ln \left(\frac{1}{\epsilon} \right)$.

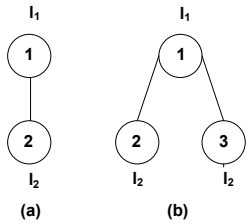


Figure 5.6: Problem with NPA

The NPA model is fast to calculate. However, there is a potential issue: For each vertex, it only considers the nearest neighbor of each label. Thus it cannot differentiate the situations when there are more than one nearest vertices with the same label. Figure 5.6 shows two graphs. In both cases, $sup(l_1, l_2) = 0.37$ according to NPA. In order to differentiate them, we propose the second model, Normalized Probabilistic Association, to take into account all the nearest occurrences of the same label.

5.5.2 Normalized Probabilistic Association

In the normalized probabilistic association model (NmPA), we try to normalize the association by the number of neighbors who have the same label.

Definition 5.6 (Normalized Association)**.** *Given an attributed unweighted graph G and a node u, if the number of neighbors of u is n and there are m neighbors having the label l, the normalized probabilistic association of l at u is $NA_u(l) = P(L(u)|l) = \frac{m}{n+1}e^{-\alpha}$.*

The normalizing factor $Z = \left(\frac{m}{n+1}\right)$ will give more association strength for the labels that are contained by many neighbors. In order to differentiate the two cases in Figure 5.6, we choose $n+1$ rather than n as the denominator. Since $NA_v(l) \leq 1$, the downward closure property is maintained. For weighted graphs, the modified version of NmPA will be discussed in Section 5.5.3.

For an itemset I, the support of I under NmPA could be calculated similar to NPA (see Definition 5.5), by following the joint distribution in Eq. 5.2. The supports in NmPA shall be smaller than those in NPA.

NmPA has two advantages over NPA. We have the following lemmas.

Lemma 5.1. *Given two nodes u and u', assume u and u' have the same number of neighbors, label $l \notin L(u), l \notin L(u')$, we have $NA_u(l) > NA_{u'}(l)$ if more neighbors of u contain l.*

Lemma 5.2. *Given two nodes u and v, assume u has more neighbors than v, label $l \notin L(u), l \notin L(v)$, we have $NA_u(l) > NA_v(l)$ if the percentage of u's neighbors that contain l is no less than that of v's.*

Lemmas 5.1 and 5.2 show that the NmPA could break the tie situations when there is an equal number of neighbors or an equal number of neighbors having l. NmPA favors the case when more neighbors contain l. These are desirable properties over NPA.

For the two substructures shown in Figure 5.6. The normalized association of l_2 at vertex 1 is $\frac{0.37}{2} \approx 0.19$ for Figure 5.6(a) and $\frac{0.37\times2}{3} \approx 0.25$ for Figure 5.6(b). Therefore, NmPA assigns a higher support for $\{l_1, l_2\}$ in Figure 5.6(b) than that in Figure 5.6(a). It can be verified that the $sup(l_1, l_2)$ values will be $1.0, 1.0, 0.59, 0.52, 0.50$ for the substructures shown in Figure 5.4(a), (b), (c), (d) and (e) respectively. Therefore, similar to NPA, the NmPA support of $\{l_1, l_2\}$ decrease gradually from Structures (a) to (e) in Figure 5.4.

Next we apply Algorithm 3, as before, to find the intermediate dataset. The only change will be in Line 8 of Algorithm 3. We shall use the following equation to update the probability,

$$NA_u(l) = \frac{1}{n+1} \sum_{v \in N(u)} e^{-\alpha} * NA_v(l), \tag{5.3}$$

where $NA_v(l)$ is the association strength of l at v and $N(u)$ is the neighbor set of u. Since l could be a label propagated from another vertex, $NA_v(l)$ could be less than 1.

Complexity. NmPA has the same complexity as NPA. However, in practice the propagation decays much faster since we normalize the probabilistic association with respect to the number of neighboring nodes at every iteration. Using NPA and NmPA, the set of all proximity patterns can be determined efficiently from the intermediate dataset, as they follow the downward closure property. In addition to NPA and NmPA, other influence models could be adapted here. As

long as the probabilistic association is calculated by Definition 5.5, the same mining algorithm could be applied.

5.5.3 Modification for Weighted Graphs

It is easy to verify that NPA and NmPA are also applicable to weighted graphs. In NPA, we consider only the nearest vertex of any label among all the neighboring vertices. Suppose, the nearest occurrence of a label l is at distance d from vertex u. Then, the NPA probabilistic association of l at u is given by $N_u(l) = e^{-\alpha \cdot d}$. If there are total n neighbors from vertex u, and among them m neighbors, each at distance d_i from u, have the label l. The NmPA probabilistic association of l at u is given by $NA_u(l) = \sum_{i=1}^{m} e^{-\alpha \cdot d_i}/(n+1)$. The procedure of generating the intermediate dataset remains the same.

5.6 Probabilistic Itemset Mining

Given an attributed graph G, the proposed information propagation models such as NPA and NmPA will generate a large set of probabilistic itemsets, whose number is equal to $|V(G)|$. Each itemset has tuples $\langle I, A_{id}(I) \rangle$, where id is the vertex id and $A_{id}(I)$ is the probabilistic association of label I to this vertex. To be consistent with the terminology used in frequent itemset mining, we also call vertex as transaction.

In the existing frequent itemset mining algorithms such as Apriori [13] and FP-growth [12, 73], the support of an itemset is the number of occurrences of all the items together in that itemset. Our problem setting is inherently different since it has to multiply the fractional

support values of all the constituent items to determine the joint support of an itemset. In the following discussion, we will first describe an algorithm to mine all the proximity patterns from the intermediate dataset generated by the NPA or NmPA model described earlier. Next, we provide an approximate version that will improve efficiency and reduce memory consumption. Finally, in addition to the support definition, we introduce an objective function to measure the "interesting-ness" of a proximity pattern and make the algorithm more efficient and effective to generate only the top-k interesting patterns.

5.6.1 Exact Mining

Algorithm 4 describes an exact mining algorithm, called pFP(Probabilistic FP-Growth). pFP is derived from FP-Growth in [73]. It first removes the infrequent 1-itemsets and constructs the FP-tree, where transactions share the same upper path if their first few frequent items are the same. We briefly introduce FP-tree here. For details, readers are referred to [73]. FP-tree is a prefix tree. The root of an FP-tree is a NULL node, since each transaction can be prefixed by a NULL item. In the original FP-growth algorithm, each node v in the tree is labeled by an item I and also associated with a count, denoted by $count(v)$, representing the number of transactions that pass through the node. At the same time, a header table is built. For an entry $(I, H(I), ptr)$ in the header table, $H(I)$ denotes the count of nodes in FP-free containing the item I and ptr records the list of nodes containing the item I. This is also known as the *side-link* of I. Now, for each frequent length-1 pattern I present in the header table, the following technique is applied. The FP-growth algorithm starts from a frequent length-1 pattern, say I, and

for each node u attached to the side-link of I, it follows the path till the root of the tree. These paths are called the conditional pattern base of I. Then, an FP-tree on this conditional pattern base (conditional FP-tree) is constructed, which acts as a transaction database with respect to I. Next, the algorithm recursively mines this resulting FP-tree to form all possible combinations of itemsets prefixed with I.

In our problem setting, labels are probabilistic and we need to multiply these probabilistic association values to determine the joint association of multiple labels. To handle probabilistic itemsets, in our algorithm, each node v in the FP-tree is associated with a bucket $\mathcal{B}(v)$ consisting of the probabilistic association values of all single items contained in that node, which is a set of tuples $\langle id : A_{id}(I) \rangle$, where v is in the side-link of item I and id is the transaction id contained in v. The buckets can be stored in the disk and accessed when the corresponding nodes are processed. As we move up the tree, the buckets corresponding to the composite itemsets in the sub-header table can be formed recursively by the intersection of the buckets of its constituent items. For example, consider an intermediate dataset given in Table 5.2. Assume, the minimum support threshold is set at 0.06. Thus, the infrequent item l_5 having support=$0.15/3 = 0.05$ can be removed first. The remaining items are then arranged in a decreasing order of their frequency as l_2, l_1, l_3, l_4. The corresponding FP-tree is depicted in Figure 5.7. Note that, although l_1 has higher support than that of l_2, it is placed below l_2 in the FP-tree, since frequency of l_1 is lower than that of l_2 in the dataset.

The exact algorithm is given in Algorithm 4. Here, $H(I)$ in the header table denotes the sum of the probabilistic association values for item I. For example, while processing l_3

transaction id	l_1	l_2	l_3	l_4	l_5
1	1	0.3	0	0	0.1
2	0.5	0.2	0.5	1	0
3	0	0.2	0.5	0	0.05

Table 5.2: Proximity Patterns: Intermediate Dataset

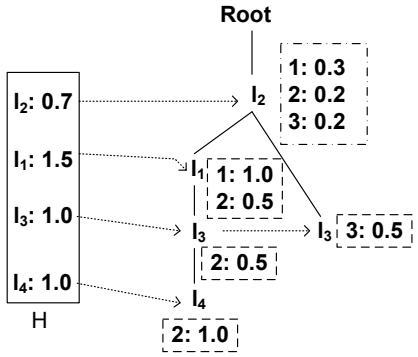


Figure 5.7: FP-Tree for Table 5.2

from the original header table H, we start moving upwards from l_3 following two distinct paths $l_3 \rightarrow l_1 \rightarrow l_2 \rightarrow root$ and $l_3 \rightarrow l_2 \rightarrow root$. The first node encountered is l_1, so it will be added in the sub-header table H_{l_3} of l_3 with $H_{l_3}(l_1) = 0.5 \times 0.5 = 0.25 > minsup \times DBSIZE$ (see Figure 5.8). So, $l_3 l_1$ is a frequent pattern. The corresponding bucket will contain only the entry $2 : 0.25$, which can be formed as an intersection of buckets of l_3 and l_1. The algorithm now recursively considers the sub-header table $H_{l_3 l_1}$ by moving upward from l_1 along the path $l_1 \rightarrow l_2 \rightarrow root$. It calculates $H_{l_3 l_1}(l_2) = 0.25 \times 0.2 = 0.05 < minsup \times DBSIZE$. So, $l_3 l_1 l_2$ is not a frequent pattern. Now, the control comes back to H_{l_3}, where the next entry is l_2, with $H_{l_3}(l_2) = 0.5 \times 0.2 + 0.5 \times 0.2 = 0.2 > minsup \times DBSIZE$. So, $l_3 l_2$ is also frequent. Its bucket will contain two entries $2 : 0.1$ and $3 : 0.1$; which can be determined by the intersection of buckets of l_3 and l_2. Note that, it cannot be extended further and this also finishes

the processing of l_3 from the original header table H. So, the algorithm starts processing l_4, which is next to l_3 in H.

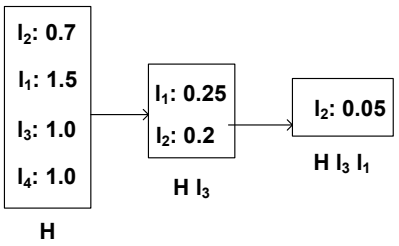


Figure 5.8: pFP applied on Table 5.2

The problem with the exact algorithm (Algorithm 4) is that, the running time increases compared to that of the original FP-growth algorithm [152], because we need to access the bucket whenever the corresponding node is processed. However, the arguments in support of this exact algorithm can be as follow.

1. Each bucket size is small compared to the original intermediate dataset size. Therefore, pFP is still efficient compared to apriori based approaches, where the whole dataset needs to be scanned every time.

2. During the execution of pFP, we need only two recent buckets in the main memory. Therefore, the buckets used before can be removed from the main memory. Since the buckets for the composite itemsets are formed by intersection of its constituent itemsets, the bucket of a large itemset usually gets smaller than that of its constituent itemsets.

5.6.2 Approximate Mining

The exact mining algorithm needs to maintain a bucket with a list of transaction ids, since we have to multiply the fractional association values to determine the joint support of multiple items. However, is it possible to compress buckets so that they can be accommodated in the main memory along with the FP-tree? The compact representation of buckets must be sufficient enough to generate an approximation to the joint association of multiple items. Here, we propose that, instead of maintaining a long bucket list of $\langle id, A_i d(l) \rangle$ for each node in the FP-tree, we can associate two variables, sum and $occurrence$, with each node.

Suppose l_x and l_y are two distinct labels appearing at nodes v_x and v_y respectively in the FP-tree, where v_x is the parent of v_y. Let, the buckets $\mathcal{B}(v_x)$ and $\mathcal{B}(v_y)$ in the exact algorithm have the association values $A_1(l_x) = x_1, A_2(l_x) = x_2, \ldots, A_n(l_x) = x_n$ and $A_1(l_y) = y_1, A_2(l_y) = y_2, \ldots, A_n(l_y) = y_n$ for transactions $1, 2, \ldots, n$ respectively. Note that some of y_i can be zero. We define sum for v_x and v_y as $sum(v_x) = \sum_{i=1}^{n} x_i$ and $sum(v_y) = \sum_{i=1}^{n} y_i$. The variable $occurrence$ is defined as the number of all non-zero occurrences of that label in the corresponding node of the FP-tree. Clearly, $occurrence(v_x) = n$ and $occurrence(v_y) \leq n$. The approximate algorithm associates these two variables sum and $occurrence$ with each node while forming the FP-tree. Now, we define the *approximate joint association* of l_x and l_y as given in Eq. 5.4.

$$\widetilde{A}(l_x, l_y) = \frac{sum(v_x) \cdot sum(v_y)}{max\{occurrence(v_x), occurrence(v_y)\}} \tag{5.4}$$
$$= \frac{1}{n} \sum_{i=1}^{n} x_i \cdot \sum_{i=1}^{n} y_i$$

The exact joint association of l_x and l_y is given by $A(l_x, l_y) = \sum_{i=1}^{n} \{x_i \cdot y_i\}$. Therefore, the absolute error E due to the approximation can be expressed as follow.

$$\begin{aligned} E &= A(l_x, l_y) - \widetilde{A}(l_x, l_y) \\ &= \sum_{i=1}^{n} [x_i \cdot (\frac{sum(v_y)}{n} - y_i)] \\ &= \sum_{i=1}^{n} [y_i \cdot (\frac{sum(v_x)}{n} - x_i)]. \end{aligned} \tag{5.5}$$

The error E is small compared to $A(l_x, l_y)$ when all the x_i's or all the y_i's are very close to each other. For example, if we consider the nodes of the FP-tree corresponding to l_2 and l_1 in Figure 5.7, the exact joint association $A = 0.40$, whereas the approximate joint association $\widetilde{A} = 0.35$, and the absolute error $E = 0.05$. Using this summarization technique, we develop aFP(approximate probabilistic FP-Growth), an approximation to pFP. Only the $build\_subtable$ procedure needs to be changed from the exact mining algorithm described earlier (see Algorithm 4). The new $build\_subtable$ procedure is given in Algorithm 5.

5.6.3 Top-k Interesting Patterns

The support value defined by our probabilistic association model only tells the association strength of a proximity pattern in a given graph. In order to measure its real "inter-

estingness", we need to compare the support value with the one generated by a randomization test.

Randomization Test. Given an attributed graph G, where each node has a set of labels, we conduct the following random permutation: Randomly select two nodes u, v and one of their labels, l_u, l_v, respectively, then swap these two labels so that u has l_v attached, and v has l_u attached. The permutation is repeated until all the labels are swapped. Let Q be the result graph.

Assume that p and q be the support value of an itemset I in G and Q respectively, using our probabilistic association model. If I is not found in the permutated graph Q, i.e., $q = 0$, we replace q with the product of support values of all its constituent labels. Now, we consider I as *interesting* if the difference between p and q is high. Note that, the higher difference between p and q indicates that the individual items in I truly formulate a pattern. If we only consider the p value of I, it might be high since some of its members occur very frequently, in which case, q value will also be high. Thus, by considering the difference between p and q, we can eliminate those uninteresting patterns from the result set. We propose to apply G-test score [139] as an objective function to measure the interestingness of a pattern.

$$p \cdot \ln \frac{p}{q} + (1-p) \cdot \ln \frac{1-p}{1-q} \tag{5.6}$$

We developed a pruning method similar to the vertical pruning approach proposed by Yan et al. [158] and integrate it with pFP and aFP to mine interesting patterns using the probabilistic FP-tree built from G and Q.

Proximity Patterns vs Frequent Itemsets. It is also possible to compare the proximity patterns mined from a graph G with the frequent itemsets mined from the node label sets if one ignores the connection between nodes. One can run the above test by replacing q with the support of frequent itemsets. The result will tell the new patterns that are missed by the classic frequent itemset mining approaches. In the experiment section, we will demonstrate such patterns.

5.7 Experimental Results

In this section, we present experimental results which illustrate the effectiveness of the information propagation model NmPA and the efficiency of our approximate itemset mining framework aFP on a number of real-life graph datasets. We are not going to experiment neighbor association model due to its time complexity. In order to evaluate the effectiveness, we report the top-k interesting patterns discovered by our approaches. We shall also analyze the effectiveness and efficiency of the approximate itemset mining algorithm (aFP) over the exact one (pFP). Finally, we provide a comparison of our result with that of frequent itemset and subgraph mining. The experiments are performed using a single core in a 32GB, 2.50GHz Xeon server.

5.7.1 Graph Datasets

Our models and mining algorithms are tested on a variety of real graph data sets including Last.FM, Intrusion network, and DBLP.

LAST.FM We crawled a local network consisting of $6,899$ users from [105]. Last.FM is a music web site where users listen to their favorite tracks and communicate with each other based on their choice of music. For each user, we crawled the most recent communications among them. These communications recommend songs. We treat them as edges. There are total $58,179$ edges. For each user, we also crawled the name of 3 artists (or musical bands) of the most recently listened tracks by that user. There are total $6,340$ artists and musical bands crawled. We mined proximity patterns among these artists and musical bands by using the social network graph that we built.

Intrusion Alert Network This network contains the anonymous log data of intrusion alerts in a computer network. It has $200,858$ nodes and $703,020$ edges where each node is a computer and an edge means a possible attack such as Denial-of-Service and TCP Service Sweep. Each node has 25 labels (computer generated alerts in this case) on average. There are around $1,000$ types of alerts. We aim to find the association of alerts in this graph data, which could reveal multi-step intrusions.

DBLP Collaboration Graph The DBLP graph is downloaded from [47]. There are $684,911$ distinct authors and $7,764,604$ collaboration edges among them. We consider the keywords present in the paper titles as the labels corresponding to these authors. We select 130 important keywords to determine the association among them. Each node has around 9 labels on average in this graph.

5.7.2 Effectiveness

#	Proximity Patterns	Score
1	Tiësto, Armin van Buuren , ATB	0.62
2	Katy Perry, Lady Gaga, Britney Spears	0.58
3	Ferry Corsten, Tiësto, Paul van Dyk	0.55
4	Neaera, Caliban, Cannibal Corpse	0.52
5	Lacuna Coil, Nightwish, Within Temptation	0.47

Table 5.3: Top-5 Proximity Patterns (Last.FM)

We present the top-5 interesting patterns for the **Last.FM** data set in Table 5.3. We applied the NmPA propagation model and the aFP mining algorithm. For the NmPA model, we set the decay constant $\alpha = 1$ and cut-off parameter $\epsilon = 0.12$. These parameters ensure that we propagate a label at most two hops. A label is propagated to a node only when at least one third of its immediate neighbors contain that label. The patterns are ranked by the G-test score defined in Eq. 5.6. Also, we report only the top-5 patterns after eliminating their smaller sub-patterns.

These patterns are practically interesting, i.e., *ATB* and *Paul van Dyk* are popular German DJ; whereas *Tiësto, Ferry Corsten and Armin van Buuren* are Dutch trance producers and DJ. *Britney Spears, Lady Gaga, Katy Perry* are American female pop singers and entertainers. *Lacuna Coil, Nightwish and Within Temptation* are Gothic metal bands from Italy, Finland and Netherlands respectively. *Neaera* and *Caliban* are death metal bands from Germany; while *Cannibal Corpse* is an American death metal band.

Table 5.4 illustrates the proximity patterns discovered by our algorithms but ranked low by the classic frequent itemset mining algorithm. It shows that the top-5 patterns in Ta-

#	Proximity Patterns	Score
1	Tiësto, Armin van Buuren , ATB	0.62
2	Katy Perry, Lady Gaga, Britney Spears	0.58
3	Ferry Corsten, Tiësto, Paul van Dyk	0.55
4	Neaera, Caliban, Cannibal Corpse	0.52
5	Lacuna Coil, Nightwish, Within Temptation	0.47

Table 5.4: Proximity Patterns minus Frequent Itemsets (Last.FM)

ble 5.3 and 5.4 are the same. That is, none of these top-5 'interesting' patterns are reported by the classical frequent itemset mining algorithm, since the items of these patterns do not co-occur frequently in individual nodes.

#	Interesting Patterns	Score
1	Ping_Sweep, Smurf_Attack	2.42
2	TFTP_Put, Audit_TFTP_Get_Filename, ICMP_Flood, Ping_Flood	2.32
3	TCP_Service_Sweep, Email_Error	1.21
4	HTML_Outlook_MailTo_Code_Execution, HTML_NullChar_Evasion	1.15
5	SQL_SSRP_Slammer_Worm, SQL_SSRP_StackBo	0.88

Table 5.5: Top-5 Proximity Patterns (Alerts)

The top-5 proximity patterns for the **Intrusion Network** data set are given in Table 5.5. The first one describes a Smurf denial of service attack. The ICMP echo request (Ping) packets addressed to an IP broadcast address cause a large number of responses, which might consume all available network bandwidth. The second one describes a TFTP (Trivial File Transfer Protocol) attack, which allows remote users to write files to the target system without any authentication. The fifth one is an attack to Microsoft SQL Server 2000 which is vulnerable to

a stack-based buffer overflow in the SQL Server Resolution Service. The discovered proximity patterns show that multiple attacks are often coupled together to complete one intrusion.

#	Interesting Patterns	Score
1	ICMP_Flood, Ping_Flood	0.94
2	Email_Error, SMTP_Relay _Not_Allowed, HTML_Null Char_Evasion	0.94
3	Image_RIFF_Malformed, HTML_NullChar_Evasion	0.90
4	TFTP_Put, Ping_Flood, Audit_TFTP_Get_Filename	0.80
5	Email_Command_Overflow, Email_Virus_Double_Extension, Email_Error	0.75

Table 5.6: Proximity Patterns minus Frequent Itemsets (Alerts)

Table 5.6 illustrates the proximity patterns discovered by our algorithms but ranked low by the classic frequent itemset mining algorithm on the intrusion network dataset. The first one is related to ICMP DOS Attack. The second one could be triggered by spammers who use an open relay to send unsolicited email to a number of email accounts. The fifth one could indicate an attacker's attempt to overflow a buffer using a command that is longer than 512 characters.

#	Interesting Patterns	Score
1	Association, Rules, Mining	1.17
2	Distributed, Network, Architecture	0.84
3	Sensor, Video, Network	0.80
4	Channel, Allocation, Network	0.67
5	Vector, Machine	0.45

Table 5.7: Top-5 Proximity Patterns (DBLP)

Table 5.7 shows the top-5 interesting patterns mined from the **DBLP** data set. Itemsets 1 and 5 are related to Data Mining and Machine Learning. Itemsets 2 is from distributed systems. The remaining patterns are from sensor and network fields.

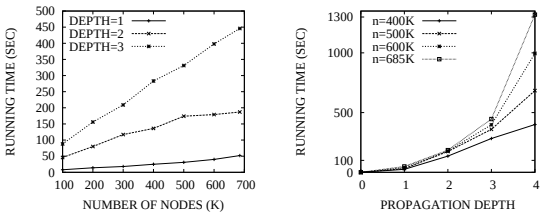
5.7.3 Efficiency and Scalability

Steps	**Last.FM**	**Intrusion**	**DBLP**
NmPA	2.0	5.0	187.0
FP-tree Formation	1.0	10.0	89.0
Top-k Pattern Mining	4.0	2.0	254.0

Table 5.8: Proximity Pattern: Runtime (sec)

We present the running time for our algorithms on the three above mentioned data sets in Table 5.8. It can be observed that each component runs pretty fast. For example, the $DBLP$ collaboration graph with about 0.7 million nodes requires less than 9 minutes to be processed.

Next, we analyze the influence of different parameters on the running time of information propagation, FP-tree building and Top-k pattern mining. We use the **DBLP** graph for these experiments. Figure 5.9(a) shows the variation of running time with respect to the number of nodes present in the graph. In order to vary the number of nodes, we randomly delete some nodes and the corresponding edges from the graph. We set the decay constant $\alpha = 1$ and vary the depth of propagation from 1 to 3 for this experiment. The cut-off parameter is set at $\epsilon = 0.36, 0.12$ and 0.04 respectively. Figure 5.9(a) shows that the $NmPA$ running time increases linearly with the increasing number of nodes. However, the slope of the lines increases as we increase the depth of propagation.



(a) # of Nodes (b) Propagation Depth

Figure 5.9: NmPA Time (DBLP)

Figure 5.9(b) shows the variation of running time with respect to the propagation depth using $NmPA$ method. We set $\alpha = 1$. In order to achieve a propagation depth at 1, 2, 3 and 4, we set the cut-off parameter $\epsilon = 0.36$, 0.12, 0.04 and 0.01 respectively. Experiments are performed for different values of n. Figure 5.9(b) shows that the NmPA running time increases exponentially with the increasing depth of propagation; which can be explained as the number of h-hop neighbors increases exponentially as we increase the depth h.

Next, we show the variation of running time with respect to the total number of labels in Figure 5.10. We use the complete **DBLP** graph and labels are selected randomly for this experiment. Similar to the previous case, we set the decay constant $\alpha = 1$ and vary the depth of propagation from 1 to 3. It can be observed that the running time of NmPA on the **DBLP** data set increases linearly with the increasing number of labels.

In Figure 5.11, we analyze the running time of FP-tree formation and top-k pattern mining using aFP with respect to the number of nodes. The propagation is done using NmPA with $\alpha = 1$ and $\epsilon = 0.12$. Note that, as we increase the number of nodes, the running time for the FP-tree formation increases almost linearly. However, the running time for mining levels off after a certain value of n. This can be explained as follow. If there are s labels, each transaction

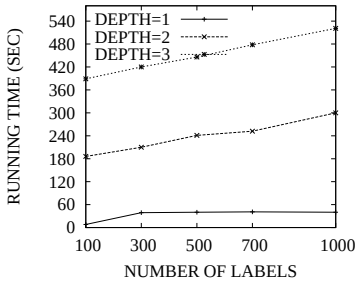


Figure 5.10: NmPA Time vs. # of Labels (DBLP)

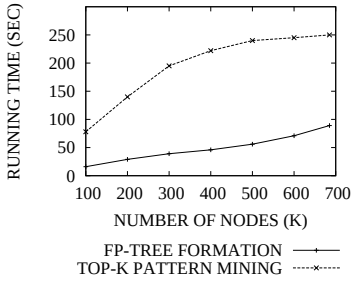


Figure 5.11: Mining Time vs. # of Nodes (DBLP)

requires at most s scans to form the complete FP-tree. So, the time complexity of building the FP-tree is almost linear in the number of nodes present in the graph. However, once the FP-tree is built, the mining depends on the size of the FP-tree and not on the actual size of the database. Hence, the running time for mining levels off after a certain value of n.

In Figure 5.12, we plot the running time of FP-tree formation and top-k pattern mining using aFP with respect to the number of labels. Note that, the latter increases at a higher rate compared to the former as we increase the number of labels.

5.7.4 Exact vs. Approximate Mining

We compare the effectiveness of our two mining algorithms, i.e. pFP (exact) and aFP (approximate) on **Last.FM** data set. Table 5.9 reports the top-5 proximity patterns minus frequent itemsets reported by the pFP mining algorithm. If we compare these patterns with those reported by the aFP in Table 5.4, the top-5 patterns remain the same. Only the score values differ slightly and therefore, the rank is a little bit different for some patterns. However, if we consider the running times given in Table 5.10, it is easy to conclude that the aFP is very efficient compared to the pFP . Moreover, this difference in running time grows very fast as the size of the database increases. For the **DBLP** graph data, the pFP mining algorithm requires about **8 hours**, whereas aFP reports the top-k patterns in less than **9 minutes**.

#	Proximity Patterns	Score
1	Katy Perry, Lady Gaga, Britney Spears	0.58
2	Ferry Corsten, Tiësto, Paul van Dyk	0.55
3	Tiësto, Armin van Buuren, ATB	0.55
4	Neaera, Caliban, Cannibal Corpse	0.51
5	Lacuna Coil, Nightwish, Within Temptation	0.46

Table 5.9: Proximity Patterns minus Frequent Itemsets using Exact Mining Algorithm (Last.FM)

Steps	aFP(approximate)	pFP(exact)
FP-tree Formation	1.0	3.0
Top-k Pattern Mining	4.0	21.0

Table 5.10: Proximity Pattern: Runtime Comparison (sec) (Last.FM)

5.7.5 Frequent Subgraph Mining

Subgraph patterns are a subset of proximity patterns, if we collapse their structure. For Last.FM, the top-5 significant patterns discovered by LEAP search [158], are given in Table 5.11. These top-5 patterns are also discovered by our probabilistic association method. If the support threshold is set at 1%, there are 67 frequent subgraphs, while our approach discovers $5,444$ proximity patterns. If we raise the support threshold further, our approach could still find interesting patterns, while the existing subgraph mining algorithm cannot. For Last.FM($\approx$ 6K nodes), the running time of subgraph mining is comparable with ours. But for the Intrusion Alert Network ($\approx$ 200K nodes), it needs about 4 hours, while our algorithm terminates within 17 seconds. Our approach avoids subgraph isomorphism testing.

#	LEAP Patterns
1	Nirvana, Arctic Monkeys, Muse
2	Radiohead, Arctic Monkeys, Muse
3	Red Hot Chili Peppers, Arctic Monkeys, Metalica
4	Radiohead, Placebo, Depeche Mode
5	Radiohead, Cold Play, Arctic Monkeys

Table 5.11: Significant Patterns via LEAP (Last.FM)

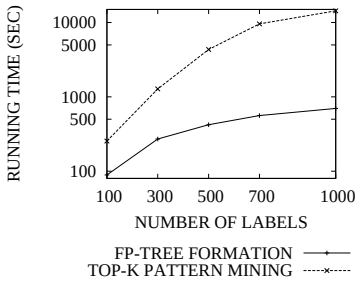


Figure 5.12: Mining Time vs. # of Labels (DBLP)

5.8 Summary

We introduced a new pattern concept in graphs - proximity pattern, which is a significant departure from the traditional concept of frequent subgraphs and frequent itemsets. Proximity pattern blurs the boundary between itemset and structure. It relaxes the rigid structure constraint of frequent subgraphs, while introducing structure association to frequent itemsets. We discussed the weakness of a neighbor association model and proposed an information propagation model that is able to transform a complex mining problem to a simplified weighted itemset mining problem, which was solved efficiently by a modified FP-tree algorithm. Furthermore, for the discovered patterns, we defined an objective function that could measure their interestingness using randomization test. In summary, we proposed a complete pipeline to define and mine novel proximity patterns in massive graphs in a scalable manner. This pipeline was evaluated on real-life social and intrusion networks. Empirical results show that it not only finds interesting patterns that are ignored by the existing approaches, but also achieves high performance for finding proximity patterns in large-scale graphs.

Algorithm 4 pFP: Probabilistic Itemset Mining

Input: Intermediate transaction dataset; and the minimum support threshold: minsup.

Output: frequent itemsets above the minsup.

Method: build the FP-tree; then call $mine\_tree(\emptyset, H)$

procedure $mine\_tree(X, H)$

1: **for all** entry I (top down order) in H **do**

2: **if** $[\frac{H(I)}{DBSIZE}] \geq$ *minsup* **then**

3: output $\{I\} \cup X$;

4: create a new header table H_I by calling $build\_subtable(I)$;

5: $mine\_tree(\{I\} \cup X, H_I)$;

6: **end if**

7: **end for**

procedure $build\_subtable(I)$

1: **for all** node v on the side-link of I **do**

2: walk up the path from v to the root once;

3: **if** encounter a node u with label J **then**

4: add/update the entry for J in H_I as below:

5: insert u as a side-link of J for that entry;

6: $H_I(J) = H_I(J) + \sum_{id \in \mathcal{B}(v) \cap \mathcal{B}(u)} \{A_{id}(J) \cdot A_{id}(I)\}$;

7: add $\{id, A_{id}(J) \cdot A_{id}(I)\}$ in $\mathcal{B}(vu)$ for all $id \in \mathcal{B}(v) \bigwedge \mathcal{B}(u)\}$;

8: **end if**

9: **end for**

Algorithm 5 aFP, Approximate Itemset Mining
procedure $build\_subtable(I)$

1: **for all** node v on the side-link of I **do**
2: walk up the path from v to root once;
3: **if** encounter a node u with label J **then**
4: add/update the entry for J in H_I as below:
5: insert u as a side-link of J for that entry;
6: calculate $\widetilde{A}(u, v)$, the approximate joint association of nodes u and v as mentioned in Section 5.6.2;
7: $H_I(J) = H_I(J) + \widetilde{A}(u, v)$;
8: $sum(vu) = \widetilde{A}(u, v)$;
9: $occurrence(vu) = occurrence(u)$;
10: **end if**
11: **end for**

Chapter 6

Conclusion and Future Directions

"In literature and in life we ultimately pursue, not conclusions, but beginnings."
Sam Tanenhaus, in *' Literature Unbound'*

6.1 Concluding Discussion

With the advent of complex social and information networks, new graph queries are emerging, including graph pattern matching and mining, similarity search, ranking, and discovery of influential nodes that require smarter and faster graph data analysis. My dissertation makes fundamental contributions in proposing effective and scalable techniques to solve these novel problems, and thereby significantly advances the state-of-the-art in this field.

In the domain of querying heterogeneous networks, we proposed NeMa [96] — a novel graph-based query-answering framework that is oblivious to the network schema. We represented the user's query also as a graph (not necessarily subgraph-isomorphic to the data graph), and defined the query results as the top-k approximate matches of the query graph in

the data graph. Our method, NeMa advances state-of-the-art network querying methods in two ways. First, we proposed a neighborhood-vectorization based cost metric for approximate subgraph matching, which relaxes the rigid structural and label matching constraints of subgraph isomorphism. Second, we designed machine learning-based efficient and scalable algorithms to identify the top-k graph matches in large networks. Empirical evaluation over several real-life datasets shows that NeMa efficiently finds high-quality matches, as compared to state-of-the-art graph querying and keyword search methods, e.g.,BLINKS [76], SAGA [142], IsoRank [138], and gStore [171]. In addition, NeMa is very robust against structural and label noises, and also scales well with the size of the data graph.

In the area of querying uncertain graphs, we proposed RQ-tree [94] — an indexing method to efficiently answer reliability queries, that is finding the set of all nodes that are reachable from a query set of nodes S with probability no less than a given threshold η. Based on RQ-tree, we defined a fast filtering-and-verification online query evaluation strategy. Extensive experiments on real-world uncertain graphs and under several settings show that our approach is very efficient—speed-up over sampling methods up to six orders of magnitude, as well as accurate—recall typically in the $[0.75, 0.98]$ range. In addition, we have shown application of RQ-tree in the influence-maximization problem [92].

In the domain of viral marketing, we designed a heuristic algorithm, RankedReplace [7] for the influence maximization problem. We then proposed a BayesTraceback model in order to approximate the effectiveness of this algorithm with the use of a faster technique. We also examined the results on a number of real social network data sets, and verified that our

methods are more effective than state-of-the-art approaches. In addition, we also considered the targeted source and targeted destination versions of the influence maximization problem.

For querying massive graph stream, we constructed a structural synopsis, called GMatrix [6], with the use of a 3-dimensional sketch structure. Our synopsis maintains information about the structural behavior of the underlying network. Thus, GMatrix is useful for a variety of structural queries such as the determination of edge frequencies, subgraph frequencies, inverse queries, or the determination of connected components. Our experimental results also show that GMatrix can compress very large streams into a small space.

In [97], we introduced a novel graph pattern, called the proximity pattern, which is a significant departure from the traditional concept of frequent subgraphs. Proximity patterns relax the rigid structure constraint of frequent subgraphs, while introducing connectivity to frequent itemsets. Therefore, it benefits from both: efficient mining in itemsets and structure proximity from graphs. We developed two models to define proximity patterns. The second one, called *Normalized Probabilistic Association* (NmPA), transforms a complex graph mining problem to a simplified probabilistic itemset mining problem, which is solved efficiently by a modified FP-tree algorithm, called pFP. NmPA and pFP were evaluated on real-life social and intrusion networks. Empirical results show that it not only finds interesting patterns that are ignored by the existing approaches, but also achieves high performance for finding proximity patterns in large-scale graphs.

6.2 Future Directions

My immediate goal is to extend my previous work on graph querying and mining, whereas my long-term goal is more open ended and related to analysis, processing and management of *BigGraphs* such as the knowledge graph.

I would like to work on the following, and other related problems, in my immediate research.

Graph Query. Can we integrate the *semantic search* within graph querying methods? Can we perform large graph alignment based on the answers of a few sample graph queries? How can one obtain diversified answers [151] over graphs, and what techniques should we use to incorporate the users' feedbacks? What features one must consider in order to build a classifier over graphs [67]? Can we leverage the *Crowdsourcing* [15, 46, 63] to formally improve the expressive power of SQL and SPARQL? Will *graph embedding* techniques [168] be useful to answer queries over social and information networks? Last, but not least, can we apply these graph querying techniques in the *web search*? We often like to get direct answers of our queries instead of links that point to various webpages. With the emergence of several types of graph-structured data, such as Freebase, DBpedia, Yago and Google's knowledge graph, integrating graph querying techniques has become an essential next step for an improved web search experience.

Uncertain Graphs. Many classical graph problems, such as *community finding* and *graph clustering* can be considered within the context of uncertain graphs. Next, how can one embed a

probabilistic graph in a lower dimension [141]? I am also interested in the influence maximization problem with *co-operative campaigners*, where each campaigner wants to maximize her influence in her native region, and does not want to influence other campaigners' native regions.

Dynamic Graphs and Streams. I would like to study graph matching and mining problems in streaming and semi-streaming models [58]. For example, how can one perform *co-clustering* over graph streams? Can we mine the *periodic proximity patterns* from graph streams and time-evolving graphs? How do we detect the *suspicious graph patterns* in a telephone or email network where the communications are received as graph streams [33]? How can one deal with dynamic updates in the query graphs?

Finally, my long-term goal is to explore new topics in the domain of *BigGraphs*. The continued growth of semi-structured and network data and novel applications – along with the emergence of cost-effective storage – ensure that the area of BigGraphs processing and management will pose many interesting problems. Usually the challenges in BigData are classified into three broad categories: *volume*, *variety*, and *velocity*. Due to variety in BigData, researchers are looking for a flexible data-model that can capture the relations among various entities, while being not strictly typed. Therefore, graph has the potential to be a data-model for BigData. One may need to consider various aspects beyond the structure of the graph, for example, the node and edge attributes, uncertainty, and graph streams. Interdisciplinary knowledge from social science, machine learning, databases, distributed computing, and graph theory are also important. I am very interested in investigating the important problems in these domains.

Bibliography

[1] http://www.blogher.com/spreading-news.

[2] (RWE) Towards Proximity Pattern Mining in Large Graphs. http://event.cwi.nl/SIGMOD-RWE/2010/07-c4de77/review-1.pdf, 2010.

[3] S. Abiteboul. Querying Semi-Structured Data. *ICDT*, 1997.

[4] L. A. Adamic and E. Adar. Friends and Neighbors on the Web. *Social Networks*, 25(3):211–230, 2003.

[5] E. Adar and C. Re. Managing Uncertainty in Social Networks. *IEEE Data Eng. Bull.*, 30(2):15–22, 2007.

[6] C. Aggarwal, A. Khan, and X. Yan. A Probabilistic Synopsis for Massive Graph Streams. In *submitted*.

[7] C. Aggarwal, A. Khan, and X. Yan. On Flow Authority Discovery in Social Networks. In *SDM*, 2011.

[8] C. Aggarwal, Y. Li, J. Wang, and J. Wang. Frequent Pattern Mining with Uncertain Data. In *KDD*, 2009.

[9] C. C. Aggarwal. *Managing and Mining Uncertain Data*. Springer, 2009.

[10] R. Agrawal, A. Borgida, and H. V. Jagadish. Efficient MAnagement of Transitive Relationships in Large Data and Knowledge Bases. In *SIGMOD*, 1989.

[11] R. Agrawal, T. Imielinski, and A. Swami. Database Mining: A Performance Perspective. *IEEE Trans. Knowledge and Data Engineering*, 5:914–925, 1993.

[12] R. Agrawal, T. Imielinski, and A. Swami. Mining Association Rules between Sets of Items in Large Databases. In *SIGMOD*, pages 69–84, 1993.

[13] R. Agrawal and R. Srikant. Fast Algorithms for Mining Association Rules. In *VLDB*, 1994.

[14] A. V. Aho, M. R. Garey, and J. D. Ullman. The Transitive Reduction of a Directed Graph. *SICOMP*, 1(2), 1972.

[15] Y. Amsterdamer, Y. Grossman, T. Milo, and P. Senellart. Crowd Mining. In *SIGMOD13*.

[16] J. R. Anderson. A Spreading Activation Theory of Memory. *J. Verbal Learning and Verbal Behavior*, 1983.

[17] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting Protein Complex Membership using Probabilistic Network Reliability. *Genome Res.*, 14:1170–1175, 2004.

[18] W. Au and K. C. C. Chan. Farm: A data mining system for discovering fuzzy association rules. In *FUZZY-IEEE*, 1999.

[19] S. N. B. Bringmann. What Is Frequent in a Single Graph? In *PAKDD*, 2008.

[20] M. O. Ball. Computational Complexity of Network Reliability Analysis: An Overview. *IEEE Tran. on Reliability*, 35:230–239, 1986.

[21] P. Barceló, L. Libkin, and J. L. Reutter. Querying Graph Patterns. *PODS*, 2011.

[22] M. Bayati, M. Gerritsen, D. F. Gleich, A. Saberi, and Y. Wang. Algorithms for Large, Sparse Network Alignment Problems. *ICDM*, 2009.

[23] H. Berger, M. Dittenbach, and D. Merkl. An Adaptive Information Retrieval System based on Associative Networks. *APCCM*, 2004.

[24] T. Bernecker, H.-P. Kriegel, M. Renz, F. Verhein, and A. Zuefle. Probabilistic Frequent Itemset Mining in Uncertain Databases. In *KDD*, 2009.

[25] S. Bharathi, D. Kempe, and M. Salek. Competitive Influence Maximization in Social Networks. In *WINE*, 2007.

[26] P. Boldi, F. Bonchi, A. Gionis, and T. Tassa. Injecting Uncertainty in Graphs for Identity Obfuscation. *PVLDB*, 5(11):1376–1387, 2012.

[27] P. Boldi and S. Vigna. The WebGraph framework I: Compression techniques. In *WWW*, 2004.

[28] C. Borgelt and M. R. Berthold. Mining Molecular Fragments: Finding Relevant Substructures of Molecules. In *ICDM*, 2002.

[29] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. *Comput. Netw. ISDN Syst.*, 30:107–117, 1998.

[30] N. Buchan and R. Croson. The Boundaries of Trust: Own and Others' Actions in US and China. *J. Econ. Behav. and Org.*, 2004.

[31] N. Buchan, E. Johnson, and R. Croson. Lets Get Personal: An International Examination of the Influence of Communication, Culture and Social Distance on Other Regarding Preferences. *Journal of Economic Behavior and Organization*, 60(3):373–398, 2006.

[32] C. Budak, D. Agrawal, and A. E. Abbadi. Limiting the Spread of Misinformation in Social Networks. In *WWW*, 2011.

[33] C. Buragohain, L. Foschini, and S. Suri. Untangling the Braid: Finding Outliers in a Set of Streams. In *ALENEX*.

[34] D. Chakrabarti, Y. Wang, C. Wang, J. Leskovec, and C. Faloutsos. Epidemic Thresholds in Real Networks. *ACM Trans. on Inf. Systems and Security*, 10(4), 2008.

[35] J. Chen, W. Hsu, M.-L. Lee, and S.-K. Ng. NeMoFinder: Dissecting Genome-Wide Protein-Protein Interactions with Meso-Scale Network Motifs. In *KDD*, 2006.

[36] W. Chen, A. Colin, R. Cumming, T. Ke, Z. Liu, D. Rincon, X. Sun, Y. Wang, W. Wei, and Y. Yuan. Influence Maximization in Social Networks when Negative Opinions May Emerge and Propagate. In *SDM*, 2011.

[37] W. Chen, C. Wang, and Y. Wang. Scalable Influence Maximization for Prevalent Viral Marketing in Large-Scale Social Networks. In *KDD*, 2010.

[38] W. Chen, Y. Wang, and S. Yang. Efficient Influence Maximization in Social Networks. In *KDD*, 2009.

[39] W. Chen, Y. Wang, and S. Yang. Efficient Influence Maximization in Social Networks. In *KDD*, 2009.

[40] J. Cheng, Y. Ke, W. Ng, and A. Lu. FG-Index: Towards verification-free query processing on graph databases. In *SIGMOD*, 2007.

[41] V. S. Cherukuri and K. S. Candan. Propagation-Vectors for Trees (PVT): Concise yet Effective Summaries for Hierarchical Data and Trees. *LSDS-IR*, 2008.

[42] E. Cohen, E. Halperin, H. Kaplan, and U. Zwick. Reachability and Distance Queries via 2-Hop Labels. *SIAM*, 32(5):1335–1355, 2003.

[43] S. Cook. The Complexity of Theorem-proving Procedures. In *STOC*, 1971.

[44] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento. A (sub)graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Tran. Pattern Anal. and Machine Int.*, 2004.

[45] S. Das, E. I. Chong, G. Eadon, and J. Srinivasan. Supporting Ontology-Based Semantic Matching in RDBMS. *VLDB*, 2004.

[46] S. B. Davidson, S. Khanna, T. Milo, and S. Roy. Using the Crowd for Top-k and Group-by Queries. In *EDBT13*.

[47] DBLP. `http://www.informatik.uni-trier.de/~ley/db/`.

[48] DBpedia. `http://dbpedia.org/About`.

[49] Delicious. `http://www.delicious.com`.

[50] M. Deshpande, M. Kuramochi, N. Wale, and G. Karypis. Frequent substructure-based approaches for classifying chemical compounds. *IEEE Trans. on Knowledge and Data Engineering*, 17:1036–1050, 2005.

[51] R. Diestel. *Graph Theory (3rd ed.)*. 2005.

[52] P. Domingos and M. Richardson. Mining the Network Value Customers. In *KDD*, 2001.

[53] Facebook. `http://www.facebook.com`.

[54] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu, and Y. Wu. Graph Pattern Matching: From Intractable to Polynomial Time. *PVLDB*, 2010.

[55] W. Fan, J. Li, S. Ma, H. Wang, and Y. Wu. Graph Homomorphism Revisited for Graph Matching. *PVLDB*, 2010.

[56] W. Fan, J. Li, X. Wang, and Y. Wu. Query Preserving Graph Compression. In *SIGMOD*, 2012.

[57] T. Feder and R. Motwani. Clique Partitions, Graph Compression and Speeding-up Algorithms. *JCSS*, 51(2):261–272, 1995.

[58] J. Feigenbaum, S. Kannan, A. McGregor, S. Suri, and J. Zhang. On Graph Problems in a Semi-Streaming Model. *Theor. Comput. Sci.*, 348(2-3):207–216, 2005.

[59] S. Feld. The Focused Organization of Social Ties. *American Journal of Sociology*, 86:1015–1035, 1981.

[60] M. Fiedler and C. Borgelt. Support Computation for Mining Frequent Subgraphs in a Single Graph. In *MLG*, pages 25–30, 2007.

[61] G. S. Fishman. A Comparison of Four Monte Carlo Methods for Estimating the Probability of s-t Connectedness. *IEEE Tran. on Reliability*, 35(2):145–155, 1986.

[62] Flickr. `http://www.flickr.com`.

[63] M. J. Franklin, D. Kossmann, T. Kraska, S. Ramesh, and R. Xin. CrowdDB: Answering Queries with Crowdsourcing. In *SIGMOD11*.

[64] Freebase. http://www.freebase.com.

[65] L. Freeman. Centrality in Social Networks: Conceptual Clarification. *Social Networks*, 1:215–239, 1979.

[66] B. Gallagher. Matching Structure and Semantics: A Survey on Graph-Based Pattern Matching. *AAAI FS.*, 2006.

[67] T. Gärtner, P. A. Flach, and S. Wrobel. On Graph Kernels: Hardness Results and Efficient Alternatives. *COLT*, 2003.

[68] J. Ghosh, H. Q. Ngo, S. Yoon, and C. Qiao. On a Routing Problem Within Probabilistic Graphs and its Application to Intermittently Connected Networks. In *INFOCOM*, 2007.

[69] A. V. Goldberg and R. E. Tarjan. A new approach to the maximum-flow problem. *J. ACM*, 35(4):921–940, 1988.

[70] J. Goldenberg, B. Libai, and E. Muller. Talk of the Network: A Complex System Look at the Underlying Process of Word-of-Mouth. *Marketing Letters*, 12:211–223, 2001.

[71] A. Goyal, W. Lu, and L. V. S. Lakshmanan. CELF++: Optimizing the Greedy Algorithm for Influence Maximization in Social Networks. In *WWW*, 2011.

[72] J. Han. Mining Heterogeneous Information Networks by Exploring the Power of Links. *ALT*, 2009.

[73] J. Han, J. Pei, and Y. Yin. Mining Frequent Patterns without Candidate Generation. In *SIGMOD*, 2000.

[74] L. Han, T. Finin, and A. Joshi. GoRelations: An Intuitive Query System for DBpedia. *LNCS*, 2011.

[75] H. He and A. Singh. Efficient Algorithms for Mining Significant Substructures in Graphs with Quality Guarantees. In *ICDM*, 2007.

[76] H. He, H. Wang, J. Yang, and P. S. Yu. BLINKS: Ranked Keyword Searches on Graphs. *SIGMOD*, 2007.

[77] hi5. `http://hi5.com`.

[78] E. Hoffman, K. McCabe, K. Shachat, and V. Smith. Social Distance and Other-Regarding Behavior. *American Economic Review*, 86:653–666, 1996.

[79] L. B. Holder, D. J. Cook, and S. Djoko. Substructure Discovery in the Subdue System. In *KDD*, 1994.

[80] M. Hua and J. Pei. Probabilistic Path Queries in Road Networks: Traffic Uncertainty aware Path Selection. In *EDBT*, 2010.

[81] J. Huan, W. Wang, D. Bandyopadhyay, J. Snoeyink, J. Prins, and A. Tropsha. Mining spatial Motifs from Protein Structure Graphs. In *RECOMB*, 2004.

[82] J. Huan, W. Wang, J. Prins, and J. Yang. Spin: Mining Maximal Frequent Subgraphs from Graph Databases. In *KDD*, 2004.

[83] IMDB. `http://www.imdb.com/interfaces#plain`.

[84] A. Inokuchi, T. Washio, and H. Motoda. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *PKDD*, 1998.

[85] J. Liu and X. Dong and A. Halevy. Answering Structured Queries on Unstructured Data. *WebDB*, 2006.

[86] N. Jayaram, M. Gupta, A. Khan, C. Li, X. Yan, and R. Elmasri. GQBE: Querying Entity Relationship Graphs by Example Tuples. In *submitted*.

[87] R. Jin, L. Liu, B. Ding, and H. Wang. Distance-Constraint Reachability Computation in Uncertain Graphs. *PVLDB*, 4(9):551–562, 2011.

[88] R. Jin, Y. Xiang, N. Ruan, and H. Wang. Efficient Answering Reachability Queries on Very Large Directed Graphs. In *SIGMOD*, 2008.

[89] M. Kargar and A. An. Keyword Search in Graphs: Finding r-cliques. *PVLDB*, 4(10):681–692, 2011.

[90] G. Karypis and V. Kumar. A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs. *SIAM J. on Scientific Computing*, 20(1):359–392, 1999.

[91] B. P. Kelley, B. Yuan, F. Lewitter, R. Sharan, B. R. Stockwell, and T. Ideker. PathBLAST: A Tool for Alignment of Protein Interaction Networks. *Nucleic Acids Res*, 2004.

[92] D. Kempe, J. Kleinberg, and E. Tardos. Maximizing the Spread of Influence through Social Network. In *KDD*, 2003.

[93] D. Kempe, J. Kleinberg, and E. Tardos. Influential Nodes in a Diffusion Model for Social Networks. In *ICALP*, 2005.

[94] A. Khan, F. Bonchi, A. Gionis, and F. Gullo. RQtree: an Index for Reliability Queries. In *submitted*.

[95] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood Based Fast Graph Search in Large Networks. In *SIGMOD*, 2011.

[96] A. Khan, Y. Wu, C. Aggarwal, and X. Yan. NeMa: Fast Graph Search with Label Similarity. In *VLDB*, 2013.

[97] A. Khan, X. Yan, and K.-L. Wu. Towards Proximity Pattern Mining in Large Graphs. In *SIGMOD*, 2010.

[98] J. W. Kim and K. S. Candan. CP/CV: Concept Similarity Mining without Frequency Information from Domain Describing Taxonomies. *CIKM*, 2006.

[99] J. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *JACM*, 46(5):604–632, 1999.

[100] N. J. Krogan, G. Cagney, and al. Global Landscape of Protein Complexes in the Yeast Saccharomyces Cerevisiae. *Nature*, 440(7084):637–643, 2006.

[101] M. Kuramochi and G. Karypis. Frequent Subgraph Discovery. In *ICDM*, 2001.

[102] M. Kuramochi and G. Karypis. Finding Frequent Patterns in a Large Sparse Graph. In *SDM*, 2004.

[103] D. L.-Nowell and J. Kleinberg. The Link Prediction Problem for Social Networks. In *CIKM*, 2003.

[104] T. Lappas, E. Terzi, D. Gunopulos, and H. Mannila. Finding Effectors in Social Networks. In *KDD*, 2010.

[105] Last.FM. `http://www.last.fm`.

[106] P. Lazarsfeld and R. Merton. Friendship as a Social Process: A Substantive and Methodological Analysis. In *Freedom and Control in Modern Society*, pages 18–66. Van Nostrand, 1954.

[107] LEDA. `http://www.algorithmic-solutions.com/leda/`.

[108] J. Leskovec, A. Krause, C. Guestrin, C. Faloutsos, J. VanBriesen, and N. Glance. Cost-effective Outbreak Detection in Networks. In *KDD*, 2007.

[109] J. Leskovec, M. McGlohon, C. Faloutsos, N. Glance, and M. Hurst. Cascading Behavior in Large Blog Graphs. In *SDM*, 2007.

[110] Z. Liang, M. Xu, M. Teng, and L. Niu. NetAlign: A Web-based Tool for Comparison of Protein Interaction Networks. *Bioinfo.*, 2006.

[111] S. Liu, L. Ying, and S. Shakkottai. Influence Maximization in Social Networks: An Ising-model-based Approach. In *Allerton*, 2010.

[112] Livejournal. `http://www.livejournal.com`.

[113] H. Ma, H. Yang, M. R. Lyu, and I. King. Mining Social Networks using Heat Diffusion Process for Marketing Candidate Selection. In *CIKM*, 2008.

[114] S. Ma, Y. Cao, W. Fan, J. Huai, and T. Wo. Capturing Topology in Graph Pattern Matching. *PVLDB*, 2012.

[115] G. Malewicz, M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, and G. Czajkowski. PREGEL: A System for Large-Scale Graph Processing. *SIGMOD*, 2010.

[116] A. Mangalampalli and V. Pudi. Fuzzy Logic-based Pre-processing for Fuzzy Association Rule Mining. Technical report, International Institute of Information Technology Hyderabad, India, 2008.

[117] S. Melnik, H. G.-Molina, and E. Rahm. Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching. *ICDE*, 2002.

[118] S. Milgram. The Small World Problem. *Psychology Today*, 6:62–67, 1967.

[119] M. Mongiovì, R. D. Natale, R. Giugno, A. Pulvirenti, A. Ferro, and R. Sharan. SIGMA: A Set-Cover-Based Inexact Graph Matching Algorithm. *J. Bioinfo. and Comp. Bio.*, 2010.

[120] J. Moody. Peer Influence Groups: Identifying Dense Clusters in Large Networks. *Social Networks*, 23(4):261–283, 2001.

[121] Myspace. `http://www.myspace.com`.

[122] NetHEPT. `http://www.arXiv.org`.

[123] M. E. J. Newman, S. Forrest, and J. Balthrop. Email Networks and the Spread of Computer Viruses. *Phys. Rev. E 66*, 2002.

[124] S. Nijssen and J. Kok. A Quick Start in Frequent Structure Mining Can Make a Difference. In *KDD*, 2004.

[125] Orkut. `http://www.orkut.com`.

[126] C. Papadimitriou and M. Yannakakis. Optimization, Approximation, and Complexity Classes. *J. Comp. and Sys. Sc.*, 1991.

[127] Y. Papakonstantinou and V. Vassalos. Query Rewriting for Semistructured Data. *SIGMOD*, 1999.

[128] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. NY: McGraw Hill, 1991.

[129] J. S. Park, M. S. Chen, and P. S. Yu. Mining Association Rules with Adjustable Accuracy. In *IBM Research Report*, 1995.

[130] J. Pearl. Reverend Bayes on inference engines: A distributed Hierarchical Approach. *American Association of AI Conf.*, 1982.

[131] M. Potamias, F. Bonchi, A. Gionis, and G. Kollios. k-Nearest Neighbors in Uncertain Graphs. *PVLDB*, 3(1):997–1008, 2010.

[132] H. T. responded To Egypt Crisis. `http://www.onlinemarketing-trends.com/2011/01/how-twitter-responded-to-egypt-crisis\_30.html`.

[133] K. Sambhoos, R. Nagi, M. Sudit, and A. Stotz. Enhancements to High Level Data Fusion using Graph Matching and State Space Search. *Inf. Fusion*, 2010.

[134] R. Schenkel, A. Theobald, and G. Weikum. Efficient Creation and Incremental Maintenance of the HOPI Index for Complex XML Documents. In *ICDE*, 2005.

[135] P. Sen, G. M. Namata, M. Bilgic, L. Getoor, B. Gallagher, and T. Eliassi-Rad. Collective Classification in Network Data. *AI Magazine*, 2008.

[136] P. Sevon, L. Eronen, P. Hintsanen, K. Kulovesi, and H. Toivonen. Link Discovery in Graphs Derived from Biological Databases. In *DILS*, 2006.

[137] H. Shang, Y. Zhang, X. Lin, and J. Yu. Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *PVLDB*, 2008.

[138] R. Singh, J. Xu, and B. Berger. Global Alignment of Multiple Protein Interaction Networks with Application to Functional Orthology Detection. *PNAS*, 2008.

[139] R. Sokal and F. Rohlf. *Biometry: the Principles and Practice of Statistics in Biological Research*. W. H. Freeman and Co., 1994.

[140] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li. Efficient Subgraph Matching on Billion Node Graphs. *PVLDB*, 2012.

[141] S. Suri. Geometric Computing over Uncertain Data. In *ALGOSENSORS*, 2012.

[142] Y. Tian, R. McEachin, C. Santos, D. States, and J. Patel. SAGA: A Subgraph Matching Tool for Biological Graphs. *Bioinfo.*, 2006.

[143] Y. Tian and J. M. Patel. TALE: A Tool for Approximate Large Graph Matching. *ICDE*, 2008.

[144] H. Tong, C. Faloutsos, B. Gallagher, and T. Eliassi-Rad. Fast Best-Effort Pattern Matching in Large Attributed Graphs. *KDD*, 2007.

[145] S. Trissl and U. Leser. Fast and Practical Indexing and Querying of Very Large Graphs. In *SIGMOD*, 2007.

[146] Twitter. `http://www.twitter.com`.

[147] J. R. Ullmann. An Algorithm for Subgraph Isomorphism. *J. ACM*, 1976.

[148] L. G. Valiant. The Complexity of Enumeration and Reliability Problems. *SIAM J. on Computing*, 8(3):410–421, 1979.

[149] S. J. van Schaik and O. de Moor. A Memory Efficient Reachability Data Structure through Bit Vector Compression. In *SIGMOD*, 2011.

[150] N. Vanetik, E. Gudes, and S. E. Shimony. Computing Frequent Graph Patterns from Semistructured Data. In *ICDM*, 2002.

[151] M. R. Vieira, H. L. Razente, M. C. N. Barioni, M. Hadjieleftheriou, D. Srivastava, C. T. Jr., and V. J. Tsotras. DivDB: A System for Diversifying Query Results. In *VLDB11*.

[152] K. Wang, L. Tang, J. Han, and J. Liu. Top Down FP-Growth for Association Rule Mining. In *PAKDD*, 2002.

[153] D. J. Watts, P. S. Dodds, and M. E. J. Newman. Identity and search in social networks. *Sience*, 2002.

[154] WebGraph. `http://webgraph.dsi.unimi.it`.

[155] Y.-C. Wei and C.-K. Cheng. Ratio cut partitioning for hierarchical designs. *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, 10(7):911–921, 1991.

[156] F. Wu, B. A. Huberman, L. A. Adamic, and J. R. Tyler. Information Flow in Social Groups. *Physica A*, 337:327–335, 2004.

[157] YAGO. `http://www.mpi-inf.mpg.de/yago-naga/yago/`.

[158] X. Yan, H. Cheng, J. Han, and P. S. Yu. Mining Significant Graph Patterns by Leap Search. In *SIGMOD*, 2008.

[159] X. Yan and J. Han. gSpan: Graph-Based Substructure Pattern Mining. In *ICDM*, 2002.

[160] X. Yan, P. S. Yu, and J. Han. Graph indexing: A frequent structure-based approach. In *SIGMOD*, pages 335–346, 2004.

[161] G. Yang. Computational Aspects of Mining Maximal Frequent Patterns. *Theor. Comput. Sci.*, 362(1):63–85, 2006.

[162] J. Yao, B. Cui, L. Hua, and Y. Huang. Keyword Query Reformulation on Structured Data. *ICDE*, 2012.

[163] H. Yildirim, V. Chaoji, and M. J. Zaki. GRAIL: Scalable Reachability Index for Large Graphs. In *VLDB*, 2010.

[164] M. J. Zaki. Scalable Algorithms for Association Mining. *IEEE Trans. Knowledge and Data Engineering*, 12:372–390, 2000.

[165] S. Zampelli, Y. Deville, C. Solnon, S. Sorlin, and P. Dupont. Filtering for Subgraph Isomorphism. *CP*, 2007.

[166] S. Zhang, J. Yang, and W. Jin. SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. *PVLDB*, 2010.

[167] P. Zhao, J. Yu, and P. Yu. Graph Indexing: Tree + Delta >= Graph. In *VLDB*, 2007.

[168] X. Zhao, A. Sala, C. Wilson, H. Zheng, and B. Y. Zhao. Orion: Shortest Path Estimation for Large Social Graphs. In *WOSN10*.

[169] Q. Zhong, H. Li, J. Li, G. Xie, J. Tang, L. Zhou, and Y. Pan. A Gauss Function Based Approach for Unbalanced Ontology Matching. *SIGMOD*, 2009.

[170] K. Zhu, W. Zhang, G. Zhu, Y. Zhang, and X. Lin. BMC: An Efficient Method to Evaluate Probabilistic Reachability Queries. In *DASFAA*, 2011.

[171] L. Zou, J. Mo, L. Chen, M. T. Özsu, and D. Zhao. gStore: Answering SPARQL Queries via Subgraph Matching. *VLDB*, 2011.

[172] Z. Zou, H. Gao, and J. Li. Discovering Frequent Subgraphs over Uncertain Graph Databases under Probabilistic Semantics. In *KDD*, 2010.