Similarity Search over Network Structure

Fan Wang

Doctor of Philosophy

December 2021

Dedication

To my parents

Aston University

Similarity Search over Network Structure

Fan Wang

Doctor of Philosophy

December 2021

With the advent of the Internet, graph-structured data are ubiquitous. An essential task for graph-structured data management is similarity search based on graph topology, with a wide spectrum of applications, *e.g.,* web search, outlier detection, co-citation analysis, and collaborative filtering. These graph topology data arrive from multiple sources at an astounding velocity, volume and veracity. While the scale of network structured data is increasing, existing similarity search algorithms on large graphs are impractical due to their expensive costs in terms of computational time and memory space. Moreover, dynamic changes (*e.g.,* noise and abnormality) exists in network data, and it arises from many factors, such as data loss in transfer, data incompleteness, and dirty reading. Thus, the dynamic changes have become the main barrier to gaining accurate results for efficient network analysis.

In real Web applications, CoSimRank has been proposed as a robust measure of node-pair similarity based on graph topology. It follows a SimRank-like notion that "two nodes are considered as similar if their in-neighbours are similar", but the similarity of each node with itself is not constantly 1, which is different from SimRank. However, existing work on CoSimRank is restricted to static graphs. Each node pair CoSimRank score is retrieved from the sum of dot products of two Personalised PageRank vectors. When the graph is updated with edges(nodes) addition and deletion over time, it is cost-inhibitive to recompute all CoSimRank scores from scratch, which is impractical. RoleSim is a popular graph-structural role similarity search measure with many applications (*e.g.,* sociometry), it can get the automorphic equivalence of nodes pair similarity, which SimRank and CoSimRank lack. But the accuracy of RoleSim algorithm can be improved. In this study, (1) we propose fast dynamic scheme, D-CoSim and D-deCoSim, for accurate CoSimRank search over large-scale evolving graphs. (2) Based on D-CoSim, we also propose fast scheme, F-CoSim and Opt_F-CoSim, which greatly accelerates CoSimRank search over static graphs. Our theoretical analysis shows that D-CoSim, D-deCoSim F-CoSim and Opt_F-CoSim guarantee the exactness of CoSimRank scores. Experimental evaluations verify the superiority of D-CoSim and D-deCoSim over evolving graphs, and the fast speedup of F-CoSim and Opt_F-CoSim on large-scale static graphs against its competitors, without any loss of accuracy. (3) We propose a novel role similarity search algorithm FaRS, and a speedup algorithm Opt_FaRS, which guarantees the automorphic equivalence capture, and captures the information from the neighbour's class. The experimental results of FaRS and Opt_FaRS show that our algorithms achieves higher accuracy than baseline algorithms.

Keywords: Social Network, SimRank Model, Role-based similarity, Web search

Acknowledgements

First and foremost, I want to thank my supervisors, Dr. Hai Wang and Dr. Weiren Yu, for providing me with invaluable direction and strength throughout my PhD studies. Their intelligent suggestions and unwavering information sharing constantly motivate me to improve my work. Nothing in this small space can adequately express my gratitude and admiration for them.

I am grateful to all members of the computer science group, especially Dr. Hongxia (Helen) Wang, who has encouraged me to continue working on the original concept of this project. I'm particularly grateful for the advise and support I received from the School of Engineering and Applied Sciences' research and administrative employees during my PhD studies.

I also want to express my gratitude to all of my friends and colleagues at Aston University and elsewhere. My PhD research work could not have been completed in the current shape without their encouragement and assistance. I'm afraid I will not be able to mention all of their names here. I am grateful to my friends Rui Zhu, Dr. Xi He, and Dr. Lei He for their constant support and friendly company. Thank you so much to Lin Gui, Nhat Do, Vishwash Batra, and Gabriele Pergola for making our time together so delightful.

Finally, I am grateful to my family: my parents for their unending love and support, as well as my grandparents and extended family members for their tremendous care and attentive attention in my life.

Thanks to all of you

Contents

Dedication **2**

Abstract **3**

Acknowledgements **5**

Contents **6**

List of Tables **9**

List of Figures **10**

List of Abbreviations **14**

List of Symbols **15**

Declaration **17**

1 Introduction **18**

1.1 Background 18

1.2 Applications and Motivations 20

1.3 Challenges and Contributions 21

1.3.1 Efficient Similarity Algorithms on Evolving Graphs 21

1.3.2 Similarity Detection on Large-scale Graphs 22

1.3.3 Role Similarity Detection on Social Networks 24

1.4 Thesis Outline 25

2 Literature review **26**

2.1 Similarity Search over Static Graph 26

2.1.1 Iterative Algorithms 27
2.1.2 Non-iteration Algorithm 29
2.1.2.1 Monte Carlo Sampling 30
2.1.2.2 Matrix-Based Methods 31
2.2 Similarity Search over Evolving Graphs 32
2.2.1 PageRank on Evolving Graphs 32
2.2.2 SimRank on Evolving Graphs 33
2.3 Role-based Network Analysis 34

3 Fast and Accurate Similarity Search For Evolving Graphs 36
3.1 Introduction 37
3.1.1 Motivation 39
3.1.2 Chapter Outlines 40
3.2 Preliminaries 41
3.2.1 Jeh and Widom's SimRank Model 42
3.2.2 CoSimRank Model 43
3.3 Similarity Search for Large-scale Evolving Graph Streams 44
3.3.1 The Similarity Search algorithm For Incremental Dynamic Graphs 44
3.3.1.1 Updating Adjacency Matrix 44
3.3.1.2 D-CoSim Algorithm 47
3.3.2 Similarity Search over Decremental Graph 55
3.3.2.1 Similarity Search over Decremental Dynamic Graphs with Edges' Deletion 56
3.3.2.2 Similarity Search over Decreased Dynamic Graph with Node Deletion 61
3.3.3 Combinatorial Algorithms For Similarity Search Over Dynamic Graphs . 76
3.4 Experimental Evaluation 80
3.4.1 Experimental Setting 81
3.4.2 Experimental results 82
3.4.2.1 Experimental Results of D-CoSim For Incremental Dynamic Graphs 83

3.4.2.2 Experimental Evaluation of D-deCoSim Algorithm over Decremental Dynamic Graphs . . . 86
3.5 Related Works . . . 91
3.6 Conclusion . . . 93

4 Scalable Similarity Search over Large-Scale Graphs 94
4.1 Introduction . . . 94
4.1.1 Motivation . . . 95
4.1.2 Chapter Outlines . . . 98
4.2 Why Spanning Polytree is Irreplaceable . . . 99
4.3 Computing CoSimRank Scores of Spanning Polytrees . . . 102
4.4 F-CoSim over Large-Scale Network . . . 105
4.5 Further Efficiency Improvement . . . 108
4.5.1 The Optimisation of Spanning Polytree Extraction. . . . 109
4.5.2 Efficient Single-Source CoSimRank Scores' Retrieval of Spanning Polytree. 112
4.5.3 Parallel Computing . . . 114
4.6 Experimental Evaluation . . . 118
4.6.1 Experimental Setting . . . 118
4.6.2 Experimental Results . . . 119
4.6.2.1 Experimental Results of F-CoSim . . . 119
4.6.2.2 Experimental Results of the Optimisation F-CoSim Algorithm . 123
4.7 Related Work . . . 127
4.8 Conclusion . . . 129

5 An Axiomatic Role Similarity Search Over Networks 131
5.1 Introduction . . . 132
5.1.1 Motivation . . . 134
5.1.2 Chapter Outlines . . . 140
5.2 Preliminary . . . 140
5.3 Proposed Schema . . . 143
5.3.1 FaRS: A Role-Based Similarity Algorithm Over Graphs . . . 143

5.3.2 Sec-RoleSim: Role-Based Similarity Search algorithm Based On Graph Topology . . . 147
5.3.3 The Axiomatic Properties of Sec-RoleSim . . . 149
5.4 EFFICIENT COMPUTATION . . . 154
5.4.1 Pruning Approach . . . 154
5.4.2 **P-speedup** approach . . . 160
5.4.3 **Out-speedup** approach . . . 161
5.5 The Application of Opt_FaRS Over Dynamic Graphs . . . 166
5.6 Experimental Evaluation . . . 168
5.6.1 Experimental Settings . . . 168
5.6.2 Experimental Results . . . 170
5.7 Related Work . . . 175
5.8 Conclusion . . . 178

6 Conclusions and Future Work **179**
6.1 Thesis Summary . . . 179
6.2 Future Avenues . . . 181
References . . . 183

Appendices **193**

A Configuration METIS **194**
A.1 Resource Acquisition . . . 194
A.2 Compile METIS algorithm In MATLAB . . . 195
A.3 Conclusion . . . 198

B SNAP Compilation **199**

List of Tables

2.1 Time Complexity Comparisons on Iteration Algorithms 29

2.2 Comparisons on Monte Carlo Methods 31

3.1 Three Cases of Dynamic Graphs with Node Deletion 63

3.2 Description of Datasets 81

5.1 Description of the Five Datasets 169

A.1 Basic Information of Configuration METIS 194

B.1 Basic Information of SNAP Compilation 200

B.2 Code set of SNAP Compilation 201

List of Figures

1.1 Facebook social Network 19

3.1 Dynamic Graph 36

3.2 Decomposing G into a spanning polytree T and ΔG ($= G \ominus T$) 45

3.3 Updating the Adjacency Matrix of the Incremental Dynamic Graph G 47

3.4 Edge Deletion ΔG of Graph G 57

3.5 Updating the Adjacency Matrix of Dynamic Graph G With Edges Deletion . . . 57

3.6 Case 1: Nodes Deletion ΔG of Graph G 65

3.7 Case 2 of Decremental Dynamic Graphs with Node Deletion 68

3.8 Case 3: Node Deletion ΔG of Graph G 72

3.9 4 Cases: Node Deletion ΔG of Graph G 74

3.10 Example of old web graph G(solid arrows) updated by ΔG 78

3.11 Parameter Testings 83

3.12 Time Efficiency on Incremental Graphs 83

3.13 Time Efficiency on Incremental Graphs: Varied $|\Delta G|$ for D-CoSim 84

3.14 Memory Efficiency & Scalability of D-CoSim 85

3.15 Accuracy of D-CoSim 85

3.16 Time Efficiency on Decremental Graphs (Edge deletion) 86

3.17 Time Efficiency on Decremental Graphs (Node Deletion) 87

3.18 Time Efficiency of All 3 Cases in D-deCoSim (Node) 88

3.19 Memory Efficiency & Scalability of D-deCoSim(Edge) 89

3.20 Memory Efficiency & Scalability of D-deCoSim(Node) 90

3.21 Accuracy of D-deCoSim 91

4.1 Extract a spanning tree and spanning polytree from the undirected G 101

4.2 Extract a spanning tree and spanning polytree from the directed G 101

4.3 Decompose G into a spanning polytree T and ΔG ($= G \ominus T$) 104

4.4 Comparison of Opt_Find_Spanning_PolyTree and Find_Spanning_PolyTree Algorithm 110

4.5 Parallel Computing 117

4.6 Time Efficiency on Static Graph 120

4.7 Time Efficiency: Phases in F-CoSim 120

4.8 Memory Efficiency & Scalability of F-CoSim 121

4.9 Memory Efficiency: Phases in F-CoSim 121

4.10 Memory Efficiency: Phases in F-CoSim 122

4.11 Time Efficiency Comparison of Procedure 1 and Procedure 2 123

4.12 Time Efficiency Comparison of P2-$\mathbf{S}(T)$ and Opt_P2-$\mathbf{S}(T)$ 124

4.13 Time Efficiency Comparison of F-CoSim and Opt_F-CoSim 125

4.14 Memory Efficiency Comparison of P2-$\mathbf{S}(T)$ and Opt_P2-$\mathbf{S}(T)$ 126

4.15 Memory Efficiency Comparison of F-CoSim and Opt_F-CoSim 126

4.16 Accuracy of F-CoSim and Opt_F-CoSim 127

5.1 A Social Network of A Project Studio With Three Levels (Level 1: project manager/associate project manager; Level 2: senior staffs; Level 3: junior staffs). . . 133

5.2 In-Neighbour Matrix of Node-Pair (u, v) 136

5.3 The role similarity search results of RoleSim and FaRS 137

5.4 Left side: Example of Graph G. Right side: Multi-Hop Backward Tracking Path of the Graph G 155

5.5 Iterated Computation of $\mathbf{CP}_3^2(\mathbf{out}, e)$ 159

5.6 The Maximum Matching Result of $\mathbf{CP}_4^1$ 165

5.7 Left side: example of old graph G (solid arrows) updated by $\widetilde{G}$ (new edge in dashed arrow and deleted edge with red cross). Right side:the partition of updated edges 167

5.8 FaRS with different λ 171

5.9 FaRS with different Γ 172

5.10 Accurate Evaluation of Algorithms Over Four Departments' Networks 173

5.11 Accurate Evaluation of the Opt_FaRS Algorithm on EU(K-means Clustering Method)174

5.12 Accurate Evaluation of the Opt_FaRS Algorithm (NDCG Method) 175

5.13 Time Efficiency of Algorithms 176

A.1 Online Source of METIS 195
A.2 The details of the metismex-master file 196
A.3 Generate Visual Studio project by CMake (Phase 1) 196
A.4 Generate Visual Studio project by CMake (Phase 2) 196
A.5 Redefinition in Several Header Files of METIS (Phase 1) 197
A.6 Redefinition in Several Header Files of METIS (Phase 2) 197
A.7 Environment seeting 197
A.8 Build MEX-function in MATLAB 198
A.9 The code of MEX-file comply in MATLAB 198

B.1 Comply SNAP with Cygwin 199
B.2 Configure SNAP in Visual Studio 200
B.3 Add the location of SNAP into VC++ directories 202
B.4 Add SNAP head file into project 202
B.5 The Result of SNAP Example File 203

List of Abbreviations

AS	as-735 data which is a communication graph representing autonomous systems
BFS	breadth-first search (Mehlhorn & Sanders, 2008)
CSR	CoSimRank algorithm (Rothe & Schütze, 2014)
CSM	CoSimMat algorithm (Yu & McCann, 2015a)
DFS	depth-first search (Mehlhorn & Sanders, 2008)
EE	email-EuAll data which is an EU email contact graph.
EU	email-Eu-core-temporal data from a prominent European research organisation was used to create the network.
HP	ca-HepPh data which is a collaboration graph obtained from the arXiv High Energy Physics
LJ	soc-LiveJournal data which is a social community network
NDCG	normalised discounted cumulative (Y. Wang, Wang, Li, He, & Liu, 2013)
PPR	Personalized PageRank (Brin & Page, 1998)
RTG	random typing generator (Akoglu & Faloutsos, 2009)
RWR	Random Walk with Restart (Yu & McCann, 2016)
SVD	Singular value decomposition
WG	web-Google data which is a Google web graph, where each node is a web page and each edge a hyperlink
WT	wiki-Talk data which is a Wikipedia network

List of Symbols

G	given (old) graph G
ΔG	update graph to (old) graph G
$n/\tilde{n}$	number of nodes in old/new graph
$m/\tilde{m}$	number of edges in old/new graph
$\deg_i^-$	in-degree of node i in (old) graph G
$\deg_i^+$	out-degree of node i in (old) graph G
d	average degree of (old) graph G
C	damping factor ($0 < C < 1$)
K	number of iterations
$\mathbf{A}/\tilde{\mathbf{A}}$	old/new column-normalised adjacency matrix
$\Delta\mathbf{A}$	the difference between the adjacency matrix of old graph and new graph
$\mathbf{S}/\tilde{\mathbf{S}}$	old/new SimRank matrix
$\mathbf{I}$	$n \times n$ identity matrix
$\mathbf{e}_i$	$n \times 1$ unit vector with only a 1 in i-th entry
$\mathbf{X}^T$	transpose of matrix $\mathbf{X}$
$\mathbf{I}_a$	in-neighbour set of node a from graph G
$\mathcal{D}_{out}$	. the node which in-degree of deleted node's out-neighbours is larger than 1
$\mathcal{R}$	the remain nodes set of new graph $\widetilde{G}$

$\mathbf{V}$	the node set of the graph G
$\mathbf{E}$	the edge set of the graph G
q/Q	the query/the query set
T	a spanning polytree
$\mathcal{M}(i,j)$	the maximum matching value of in-neighbour matrix of node pair (i,j)
$maxdeg^{-}(i,j)$	the maximum value of in-degree of node pair (i,j)
$maxdeg^{+}(i,j)$	the maximum value of out-degree of node pair (i,j)
$mindeg^{-}(i,j)$	the minimum value of in-degree of node pair (i,j)
$mindeg^{+}(i,j)$	the minimum value of out-degree of node pair (i,j)
λ	the normalization coefficient $(0 \leq \lambda \leq 1)$
$\mathcal{I}(u)$	the in-neighbours set of node u
$\mathcal{O}(u)$	the out-neighbours set of node u
$\mathbf{p^l}/\mathbf{P}$	the tracking path
$\mathcal{M}[\mathbf{CP}_{k-1}]$	the maximum matching result of $\mathbf{CP}_{k-1}$
$\widetilde{\mathcal{M}}[\mathbf{CP}_{k-1}]$	the matched values set of maximum matching on $\mathbf{CP}_{k-1}$
$\mathcal{B}_{ij}$	the in-neighbour matrix of node-pair (i,j)
$\mathcal{M}[\mathcal{B}_{ij}]$	the maximum matching result of $\mathcal{B}_{ij}$
$\widetilde{\mathcal{M}}[\mathcal{B}_{ij}]$	the matched values set of maximum matching on $\mathcal{B}_{ij}$
$\mathcal{M}[\mathcal{B}_{ij}, \mathbf{CP}_{k-1}]$	the matched values of $\mathbf{CP}_{k-1}$ in $\mathcal{B}_{ij}$
$\lvert\mathcal{M}_k[\mathcal{B}_{ij}, \mathbf{CP}_{k-1}]\rvert$	the number of matched values of $\mathbf{CP}_{k-1}$ in $\mathcal{B}_{ij}$

Declaration

The thesis entitled '*Similarity Search over Network Structure*' submitted for the degree of PhD in Computer Science, has been composed by myself and has not been presented or accepted in any previous application for a degree. All sources of information have been acknowledged by means of references.

Portions of the work presented in this thesis have been published in the following and conference.

Yu, W., Wang, F. (2018, April). Fast Exact CoSimRank Search on Evolving and Static Graphs. In Proceedings of the 2018 World Wide Web Conference on World Wide Web (pp. 599-608). International World Wide Web Conferences Steering Committee.

Fan Wang
30 December 2021

1 Introduction

Network structure is often ubiquitous around us. Some vivid examples include social networks, web graphs, neural networks and infrastructure networks. These data carry graph topology structures and arrive from multiple sources at an astounding velocity, volume and veracity. Many proliferating application areas, such as link analysis, web mining, recommender systems and graph databases, have resulted in a growing requirement to access single-source relevance based on network structure.

1.1 Background

What is network structure (graphs)? A basic and complete network structure (graphs) (Sedgewick & Wayne, 2011) is composed of node sets and connections between nodes. Graphs are commonly used to describe and store data with a "many-to-many" relationship, which is a critical data structure. Depending on the assumptions used to update the data, graphs can be classified into several types, such as: static graphs and dynamic graphs. The static graph's nodes and edges will not change as time passes or as other factors change. Updates are present in dynamic(evolving) graphs. However, the updates in evolving graphs are known and unique, whereas the updates in uncertain graphs are not. This thesis focuses on algorithms for both dynamic and static graphs. Graphs are categorised into directed and undirected graphs based on the types of edges in the graph. The adjacency matrix and the linked list are two standard storage methods for graphs. We mainly employed the adjacency matrix in our studies.

Why is the graph structure used? Why use the graph structure to analyse datasets when there are so many different types of data representation? It is used because the graph structure displays the type of node and the interaction between node pairs unambiguously. The existing graph-based techniques, such as the shortest path method, depth-first search approach etc. can aid data analysis. For example, consider the urban road network. Each city site represents a node, and the road linking the two nodes represents an edge, resulting in a city road network. The starting point is location a, and the destination is location b. In this case, we utilise the shortest path algorithm in the road network to find the quickest route from location a to location b.

Similarity Search Based on Network Structure. As an implement of network-structured data, the similarity search has become an important and necessary research area. Generally speaking, similarity search can be split into two types: one is text-based similarity search (such as words), and the other one is link-based similarity search (including graph and network). In this thesis, we primarily concentrated on link-based similarity search.

Analysing the relationship between nodes in a network-structured data can reveal the similarity between nodes of graphs, thus similarity searching has become a fundamental issue in graph analysis. For example, Figure 1.1 is a Facebook graph. It depicts a typical social network where each node represents a user and each edge denotes the relationship between the two users. Essentially, my research aimed to identify the similarity between any two nodes in a graph (for instance, similarity search between Nick and Alice in Figure 1). While the scale of network-

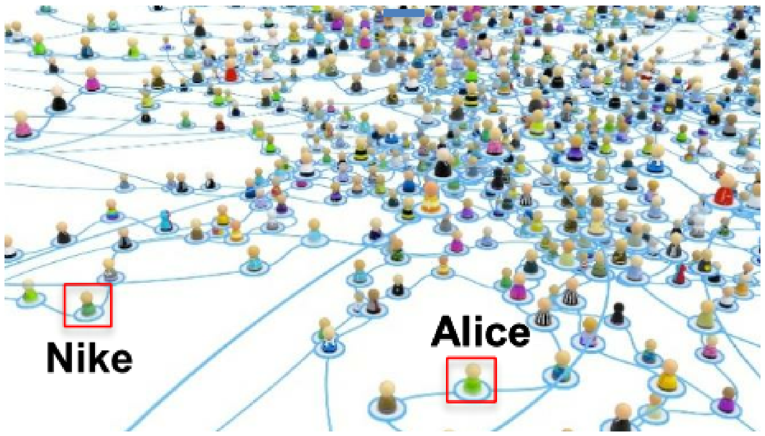


Figure 1.1: Facebook social Network

structured data is increasing, updates (such as noise and abnormality) often exist in network data. Such dynamic updates often arise from various factors, such as data loss during transfer, data incompleteness and dirty reading, and this has become the main barrier to obtaining accurate results from efficient network analysis.

1.2 Applications and Motivations

Several relevant real-world instances are provided below, which demonstrating the need for efficient algorithms to cope with similarity search on massive, dynamic network data.

Application 1 (Image Recommendation system). For image recommendation, searching for images that are similar to a given image is critical. The previous research has primarily focused on content-based similarity detection, which quantifies similarity based on visual characteristics such as colour and shape, with only a few studies paying significant attention to semantics. The semantic performance of the image recommendation system can be improved by using a similarity search algorithm on the image-tag network (M. Zhang, Yang, Dong, Wang, & Zhang, 2021). The image-tag network is made up of the relationships between images and tags, where tags and images are the nodes and the interactions between pictures and tags are the edges. The similarity search algorithm is used to provide image recommendation semantically based on the intuition that "similar images contain similar tags, and similar tags describe similar images". An image recommendation system with a similarity search algorithm can efficiently retrieve the image recommendation list with respect to the query image semantically, which is better than an image recommendation system with content similarity (M. Zhang et al., 2021). However, the number of images is constantly increasing with the continuous development of the Internet information age; thus, the content (images and tags) of images with dynamic updates. Therefore, efficient and accurate image recommendation system algorithms on big dynamic graphs are still challenging. In this thesis, we proposed a fast and exact similarity search algorithm over large-scale evolving graphs to bridge this gap(Chapter 3).

Application 2 (Community Detection). Since data from many different areas can be naturally transferred to graph typologies, networks have grown ubiquitous. Many networks can be naturally divided into many communities (K. Wu & Liu, 2021). A community is a group of comparable nodes, such as employees in an organisation with similar work or animals at the same level in the food webs. The similarity detection approach is mostly used to find communities on graph structures. Similarity search algorithms detect communities based on the intuition that "the more common the neighbour nodes of two nodes, the more similar they are". Take food webs as an example; each animal is denoted as a node, and the edge exists between prey and predators. The food web continues to expand as people's knowledge of animals grows. When dealing with the community detection of huge graphs, the existing similarity search algorithms demand a high level of computational complexity. This thesis provides the solution for enhancing the efficiency of retrieving similarity scores over large-scale static graphs(Chapter 4).

Application 3 (De-anonymisation of Social Networks). People are becoming increasingly aware of the privacy risks associated with public network information as network information continues to expand. De-anonymising the network (identifying the same person in different networks) can help it effectively assess the privacy risks of users (Shao, Liu, Shi, Zhang, & Cui, 2019). De-anonymisation is a technique for discovering information in a dataset. By comparing anonymised data with other data sources, it is possible to re-identify the source of the anonymised data. De-anonymisation can be accomplished using any information that differentiates one data source from another. The role similarity algorithm is used to achieve de-anonymisation based on the notion that correct node mappings have a high degree of similarity between nodes. To efficiently produce high-quality de-anonymisation findings, a role similarity search algorithm takes advantage of node similarities and structural information from neighbourhood matches. However, the accuracy and efficiency of existing role similarity search algorithms based on graph topology can be improved. Thus, we develop a more accurate role similarity search algorithm over graphs as well as an accelerated algorithm to improve the computation efficiency(Chapter 5).

1.3 Challenges and Contributions

The purpose of this thesis is to build unique link-based single-source similarity search models and innovative strategies for efficiently similarity searching on large-scale static and dynamic networks. Furthermore, we are interested in innovative methods for taming the computing complexity of similarity searching in a scalable manner as well as in unique effective models for enriching the semantics for role relevance assessment without incurring increased computational costs. We go over the thesis' primary obstacles.

1.3.1 Efficient Similarity Algorithms on Evolving Graphs

Real graphs are frequently huge, and links evolve with modest changes. When the graph is refreshed, reassessing the similarities of all node pairs is somewhat costly. We, therefore, examined fast and accurate similarity search algorithms on evolving networks.

SimRank (Jeh & Widom, 2002) is a well-known link-based pairwise similarity model, however, a SimRank search on vast and dynamic networks creates considerable issues due to its self-referentiality. The more efficient existing similarity search measure, CoSimRank (Rothe & Schütze, 2014), was proposed at ACL [1] 2014. Unlike the SimRank algorithm, the CoSimRank algorithm captures all meeting times of two arbitrary surfaces. The SimRank algorithm only captures the first meeting time of two random surfaces. This is why the CoSimRank algorithm retrieves more accurate similarity scores than the SimRank algorithm. However, when we apply CoSimRank over large-scale dynamic graphs, it leads to the following limitation: the CoSimRank algorithm is not sensitive to the updates of dynamic graphs. When we implement CoSimRank on dynamic graphs, it needs to compute all node pairs' similarity scores from scratch even if we only add or delete one node or edge to the old graph. Thus, it is difficult for the CoSimRank algorithm to produce quick responses over dynamic graphs due to its cost for resumptions.

We have addressed the aforementioned issue in Chapter 3. (1) We proposed a novel similarity search algorithm over large-scale incremental evolving graphs, D-CoSim. The D-CoSim algorithm can efficiently and accurately retrieve the similarity search scores of the updated parts of the dynamic graphs rather than recomputing the similarity scores of all the node pairs of the new graphs. (2) We proposed another similarity search algorithm over decremental graphs, D-deCoSim. It is similar to the D-CoSim algorithm because they are sensitive to the updated parts of the graphs. The D-deCoSim algorithm can be separated into two parts, D-deCoSim(Edge) and D-deCoSim(Node). D-deCoSim(Edge) can be implemented on the dynamic graphs with edge deletion. D-deCoSim(Node) works on the dynamic graphs with node deletion.

Using real large-scale data, we empirically verify the following: (1) For time efficiency, D-CoSim and D-deCoSim outperforms the best-known baseline algorithms by around 3–5 order-of-magnitude. (2)D-CoSim and D-deCoSim require smaller memory space than baseline algorithms to ensure memory efficiency. (3) D-CoSim and D-deCoSim retrieve similarity scores quickly and accurately on dynamic graphs, with no need to reassess them from scratch.

1.3.2 Similarity Detection on Large-scale Graphs

Graph structure plays an increasingly important role in people's lives, such as Facebook networks, online shopping networks, urban road structure networks etc. As the application of

[1] Association for Computational Linguistics Conference.

graph structure in people's lives increases, the size of graphs becomes larger and larger. Therefore, an efficient similarity detection approach that can be applied to large-scale graphs is urgent and necessary. The existing similarity search algorithms over static graphs have the following limitations: (1) Due to the limitation settings of some existing algorithms, some node pairs' similarity search scores equal 0("zero issue"). For example, SimRank (Jeh & Widom, 2002) has "zero issue". For any two nodes in the graph, if there does not exist any symmetric in-link path of node pair, the SimRank score of the node pair is 0. Similarity search algorithms with "zero issue" would miss partial information contained in the graphs. (2) For single-source similarity search over large-scale static graphs, the computation of existing algorithms is time-consuming; *e.g.,* the time complexity of CoSimRank is $O(K(m+n))$ for computing one node pair's similarity score. (3) Memory efficiency can directly affect the application range of algorithms. For example, Co-Simmate (Yu & McCann, 2015a) needs $O(n^2)$ memory to store the decomposed matrices. Thus, Co-Simmate only works on small graphs.

Motivated by this, in Chapter 4, we propose a fast and accurate similarity search model over large-scale static graphs, F-CoSim. F-CoSim is mainly composed of three phases: (i) extracting a spanning polytree from the given graph; (ii) according to the advantage of the spanning polytree, computing the similarity scores of the spanning polytree level by level; (iii) recalling the D-CoSim algorithm to generate the difference between the given graph and the spanning polytree. Furthermore, we accelerate the F-CoSim algorithm by three parts: (i) we propose a more efficient approach to extracting the spanning polytree from the given graphs; (ii) a single-source similarity search algorithm based on the spanning polytree is implemented; (iii) inspired by the definition of parallel computing, we propose a novel algorithm to speed up the F-CoSim algorithm. The first step of the accelerate algorithm is to separate graphs by following the METIS (Kusumoto, Maehara, & Kawarabayashi, 2014) approach and to compute the similarity scores of each part simultaneously. Then, it uses D-CoSim to retrieve the similarity scores of the cut edges.

Our empirical evaluations verify the following: (1) our algorithms' time efficiency performance is about 9.8 times better than the best-known baseline algorithms; (2) our algorithms retain comparable linear memory and scale on million-node graphs; (3) our algorithms improve computational efficiency without sacrificing accuracy.

1.3.3 Role Similarity Detection on Social Networks

People's social networks have grown significantly in the last decade due to the rapid advancement of electronic communication technologies. Role similarity detection is essential for analysing real complex social networks, as it can retrieve structural information from networks. RoleSim (Rothe & Schütze, 2014) is the best-known role similarity search algorithm over networks. The RoleSim algorithm was founded on the recursive philosophy that "two nodes have the same role if they interact with equivalent sets of neighbours". RoleSim retrieves the role similarity of node pairs based on graph automorphism, and the role similarity scores are computed using the maximum matching approach. However, the RoleSim algorithm has the following limitations: (1) (accuracy) the RoleSim scores are calculated using the maximum matching approach; thus, the algorithm only capture partial information from a node pair's in-neighbour matrix. The missing information causes the accuracy of the algorithm to decrease; (2) (computation efficiency) it is time-consuming to implement the RoleSim algorithm for single-source role similarity scores computation with respect to the query as redundant calculations; (3) (dynamic) if the graph constantly undergoes small changes over time when using RoleSim on dynamic networks, all node-pair role similarity scores must be computed from scratch. As a result, the RoleSim algorithm struggles to generate speedy replies over dynamic graphs due to the high cost of resumptions.

In Chapter 5, we propose a more accurate single-source role similarity search algorithm based on graph topology. Our role similarity search algorithm computes role similarity scores with respect to a query by top Γ maximum matching. The convergence, uniqueness, symmetry, boundedness and triangular inequality of our role similarity search algorithm are proved. Furthermore, an acceleration algorithm is proposed to speed up our role similarity algorithm. The acceleration algorithm can be divided into two parts: tracking path extraction and pre-computation. Then, We implement our role similarity search algorithm over dynamic graphs, and the pre-processor is chunking the updated parts of evolving graphs into groups with the same end node. Finally, the role similarity scores of the new graph can be generated by the Opt_FaRS algorithm.

The experimental evaluations along with the theoretical proofs show the following: (1) our role similarity search algorithm is more accurate than two state-of-the-art algorithms based on graph topology; (2) the accelerate role similarity search algorithm does not sacrifice accuracy for speed.

1.4 Thesis Outline

The rest of the thesis is organised in the following chapters:

Chapter 2 In this chapter, we provide a comprehensive literature review on graph-based similarity search. According to the different standards, similarity search algorithms can be divided into different types. They can be divided into static and dynamic algorithms on evolving graphs based on the assumptions pertaining to date updates. Similarity search algorithms can also be divided into iterative and non-iterative algorithms (Monte Carlo sampling and matrix-based methods) depending on the format of the models.

Chapter 3 We propose efficient similarity search algorithms over incremental and decremental dynamic graphs in this chapter. We conduct several experiments on real datasets to demonstrate that our algorithms steadily outperform two state-of-the-art algorithms (time efficiency and memory efficiency) and that our algorithms do not scarify accuracy for speeding up.

Chapter 4 In this chapter, we propose a fast and accurate similarity search algorithm over large-scale static graphs. Furthermore, we develop an acceleration algorithm to reduce the time consumption while computing similarity scores. We conduct extensive experiments on large datasets to demonstrate that our efficient similarity search algorithm outperforms the state-of-the-art approaches on static graphs with better time and memory efficiency. At the same time, our algorithms do not scarify accuracy for speeding up.

Chapter 5 We propose an accurate role similarity search algorithm based on graph topology in this chapter. Based on our role similarity search algorithm, we also propose an optimisation algorithm. We conducte several experiments on real datasets to demonstrate that our algorithms are steadily more accurate than two baseline algorithms, and our optimisation algorithm does not scarify accuracy for speeding up.

Chapter 6 This chapter summarises the research's findings, highlights the new contributions the study has made, and addresses future research directions.

The contents of the main chapter are supplemented by Appendix A and Appendix B.

2 Literature review

In the days of information and networks, similarity detection has emerged as the most effective data analysis method. Similarity detection algorithms are extensively used in big data processing in real life, such as recommendation systems (J. Wu et al., 2021; Linden, Jacobi, & Benson, 2001; Machado, Bocek, Filitz, & Stiller, 2014; Catherine & Cohen, 2016; Hang & Singh, 2010), web mining (Kinne & Lenz, 2021; Muni Manasa & Farooq, n.d.), social networks (Kumar, Novak, & Tomkins, 2006; Garg, Gupta, Carlsson, & Mahanti, 2009; Machado et al., 2014), etc. Based on people's demand for instantaneous information acquisition, efficient and accurate similarity detection algorithms on large graphs denote a problem to be solved urgently. In a graph structure, similarity detection algorithms can be divided into different types in accordance with different application environments and requirements. According to the graphs, the algorithm can be divided into static and dynamic algorithms. As per the requirements of information searchers, it can be divided into node-pair, partial node-pairs, all node-pairs, and single-source algorithms. The different implementation methods of similarity detection algorithms can be divided into iteration, Monte Carlo sampling, and matrix-based method. In this chapter, we primarily review the existing work in detail from three directions, which is inclusive of static, dynamic and role similarity detection. This chapter provides an elaborate summary of previous works in relevance evaluation and relevant fields. In the relevant chapters, a brief comparison of earlier research will be presented along with the methodologies established in this thesis.

2.1 Similarity Search over Static Graph

The development of methods for efficient similarity search over static graphs denotes a significant research area, which is widely applied in node ranking and graph data mining (Yu & Lin, 2013; Yu, Lin, & Zhang, 2013a; Yu, Lin, Zhang, Chang, & Pei, 2014; Yu, Lin, Zhang, & McCann, 2015; Yu, Lin, Zhang, Zhang, & Le, 2012; Yu & McCann, 2014, 2015b, 2015c). In this section, studies on state-of-the-art data computational methods for similarity search over the static network structured data are reviewed. According to the varied methods for similarity search, these literature can be divided into two categories, i.e., `iterative algorithms` and `non-iterative algorithms`.

2.1.1 Iterative Algorithms

Computing similarity by iterative algorithms has been proposed as powerful method in real web applications. The following is a summary of recent studies on similarity search using iterative algorithms based on graph topology.

On the basis of an intuitive and straightforward graph-theoretic model, a general similarity iterative method called SimRank (Jeh & Widom, 2002) was proposed, which is the most adopted similarity search method. The basic intuition of SimRank is that two objects are similar if they are related to similar objects. In addition to the basic intuition, they defined any node in a graph is maximally similar to itself ($S(a, a) = 1$). The central idea of the SimRank algorithm is that the similarity score of any two nodes in a graph is computed by the average similarity scores of their in-neighbours. They assign a decay factor C (where $0 \leq C \leq 1$) to prevent the similarity score of one node and itself equal to the similarity of the node's in-neighbours (if node b and node c are in-neighbours of node a, decay factor C is used to prevent $S(a, a) = S(b, c) = 1$). Intuitively, a SimRank score $S(a, b)$ aggregates all the paths of two random surfers starting at node a and node b, and the algorithm will terminate after meeting at a common node. SimRank has been a fundamental measure for evaluating the similarity of node-pair in a graph, due to its self-referentiality, accuracy,and fast SimRank computation over static graphs pose striking challenges.

In order to solve these problems, an accurate estimate and optimization techniques for SimRank computation are suggested (Lizorkin, Velikhov, Grinev, & Turdakov, 2008a). Firstly, they proposed a method to estimate the x accuracy of SimRank algorithm. They assigned that the difference between SimRank theoretical and iteration similarity scores decrease exponentially, referred to as the accuracy gap. The accuracy of SimRank score will increase while the number of iteration goes up. Similarly, while the decay number is closer to 1, the accuracy of SimRank score will decrease. This proposition implies that it can be achieved by choosing a smaller decay factor or more iterations to guarantee a more accurate similarity score. Next, they proposed an optimization technique to expedite the SimRank algorithm. They firstly chose the essential node-pair with zero scores and node without out-neighbour, which, in turn, is used to reduce the computation of each iteration. Thereafter, the essential part of the technique is partial sums, which greatly avoids repeat computations. Finally, they assign a threshold to reduce the memory space for useless similarity scores (the values of which are close to 0). This optimization technique reduces the computation of SimRank from $O(n^4)$ to $O(n^3)$ time in the worst case scenario.

SimRank similarity algorithm was firstly represented by matrix formulation in (Yu, Zhang, Lin, Zhang, & Le, 2012a). The method successfully prevents the random surfer from the two start nodes from getting stuck by the teleportation approach. The adjacency matrix is stored by the Compressed Sparse Row(CSR) scheme, which greatly reduces the memory space of computation. Yu. et al. also developed a novel optimization method such that each iteration of similarity computing takes $O(min(n \cdot m, n^r))$ time and $O(n + m)$ space, where m denotes the number of edges in a graph.

Based on the previous research findings (Lizorkin et al. 2008), a more efficient SimRank computation method has been put forward (Yu, Lin, & Zhang, 2013b). They discovered that it is possible for similarity scores across in-neighbours to have common sub-summations following partial sums. The novel clustering method of optimizing partial sum sharing eliminates redundancy computation based on minimum spanning tree. This efficient algorithm drastically reduces the time complexity of SimRank from $O(n^4)$ to $O(Kd'n^2)$, where d' is much smaller than the average in-degree of a graph. Thus, SimRank can be applied in a wide area by this measure.

Another efficient algorithm was proposed based on a matrix formulation, called CoSimRank (Rothe & Schütze, 2014). It recursively follows the SimRank-like philosophy that two nodes are considered similar if their in-neighbours are similar. However, two different key points are known to exist between SimRank and CoSimRank algorithm. Firstly, for CoSimRank, the similarity between a node and itself is not equal to 1 ($S(a, b) \neq 1$). Secondly, each CoSimRank score needs to aggregate all meeting times of two random surfers starting at the node-pair. Thus, CoSimRank has been found to be more accurate and practical than SimRank in many real-life applications. The most major improvement of CoSimRank is that the algorithm can compute the similarity score of a single node pair without computing the similarity scores of the entire graph, thus greatly improving the time complexity to $O(n^2)$.

Time Complexity Analysis Depending on the different characteristics of iteration similarity search algorithm over static graph, we summarize time complexity using three different aspects:

- **Single-pair.** The SimRank score computation $S(a, b)$ of one node pair (a, b) of the given graph.
- **All-pairs.** The SimRank scores computation $S(*, *)$ of all node pair of the given graph.
- **Per-iteration.** The SimRank scores computation $S(*, *)$ of all node pair of the given graph at each iteration.

Table 2.1: Time Complexity Comparisons on Iteration Algorithms

Literature	Type	Time Complexity
(Jeh & Widom, 2002)	All-pairs	$O(n^4)$
(Lizorkin et al., 2008a)	All-pairs	$O(n^3)$
(Yu, Zhang, et al., 2012a)	Per-iteration	$O(min(n \centerdot m, n^r))$
(Yu et al., 2013b)	All-pairs	$O(Kd'n^2)$
(Rothe & Schütze, 2014)	Single-pair	$O(n^2)$

From Table 1, it can be seen that the SimRank scores' computation became increasingly efficient along with the order of techniques. The most efficient one is CoSimRank algorithm, which needs time complexity $O(n^2)$ to compute single-pair nodes. This is because it computes a single node similarity without the similarity scores of the entire graph. In the next sub-section, we will survey state-of-art non-iteration similarity search algorithms, and divide the previous studies in accordance with different methodologies (Monte Carlo Sampling and Matrix-Based Method).

2.1.2 Non-iteration Algorithm

Retrieving similarity scores by the iterative method suffers from some limitations, such as inefficiency. In particular, when applied on an extensive scale network structured data, the iterative algorithm will entail high time complexity. Thus, there are several types of research studies on non-iteration similarity search algorithms over graphs. In this section, we divide non-iteration similarity search algorithms on graphs into two groups: Monte Carlo sampling method and matrix-based method.

2.1.2.1 Monte Carlo Sampling

Unlike the iterative similarity search algorithms, similarity search algorithms with Monte Carlo sampling methods are fast and scalable. Existing works on Monte Carlo sampling for similarity search based on graph topology include (Fogaras & Rácz, 2005; Sarlós, Benczúr, Csalogány, Fogaras, & Rácz, 2006; Tian & Xiao, 2016; Kusumoto et al., 2014).

In order to scale similarity search algorithms on large-scale graphs, Fogaras et al. proposed a Monte Carlo similarity search algorithm (Fogaras & Rácz, 2005). This algorithm refers to a link-based similarity search method, which then analyses the "proximity" of nodes in a graph for the purpose of link prediction. The algorithm is premised on the second definition of Monte Carlo methods that need to calculate the expected value of samples. When computing the expected value, this algorithm maintains all positions of random walks by a fingerprint tree structure, thus greatly enhancing the efficiency of time and space. Therefore, this algorithm is the first one to require linear time and space to compute single-pair nodes similarity scores. However, when the algorithm is applied on a graph with billions of nodes, it will entail high consumption time.

Then, Kusumoto et al. (Kusumoto et al., 2014) proposed a more efficient Mont Carlo method for computing similarity on certain graphs. This algorithm uses hash tables to maintain the position of random walks while they compute the expected value. In contrast to the Fogaras et al.s (Fogaras & Rácz, 2005) algorithm that needs to maintain all random walks, this algorithm is only required to maintain t-th random walk positions. In turn, this improves the algorithm's time efficiency. When doing top-k similarity searching, this algorithm takes $O(n)$ time, and $O(m)$ space.

The sampling approach, SLING (Tian & Xiao, 2016), is the best-of-breed SimRank algorithm on static graphs. However, their techniques, if applied to CoSimRank, are not fast because the performance gain of SLING is predicated on aggregating "only the *first* meeting time" of two coalescing walks, as opposed to CoSimRank that aggregates "*all* their meeting times".

Complexity Comparison.

Similarity search algorithms based on Monte Carlo methods have drastically enhanced the computing efficiency of computing SimRank score over large-scale graphs. We compare the three different algorithms from three aspects: time complexity (single-pair search and single-source search), space requirement, and pre-processing time (shown in Table 2.2).

Table 2.2: Comparisons on Monte Carlo Methods

Algorithm	Time Complexity		Space	Pre-processing
	Single Pair	Single Source	Requirement	Time
Fogara et al.05	$O(\log\frac{1}{\varepsilon}\log\frac{n}{\delta}/\varepsilon^2)$	$O(n\log\frac{1}{\varepsilon}\log\frac{n}{\delta}/\varepsilon^2)$	$O(n\log\frac{1}{\varepsilon}\log\frac{n}{\delta}/\varepsilon^2)$	$O(n\log\frac{1}{\varepsilon}\log\frac{n}{\delta}/\varepsilon^2)$
Kusumoto et al.14	$\ll O(n)$	$\ll O(n^2)$	$\ll O(m)$	
Tian et al.16	$O(\frac{1}{\varepsilon})$	$O(\frac{n}{\varepsilon})$	$O(\frac{n}{\varepsilon})$	$O(\frac{m}{\varepsilon}+n\log\frac{n}{\delta}/\varepsilon^2)$

We can see from Table 2.2 that the time and space complexity of SLING algorithm is near-optimal. While the time and memory efficiency of similarity search algorithms based on Monte Carlo method have been improved greatly, all these algorithms sacrifice the SimRank score accuracy for speed. Next, we will review several similarity search algorithms based on matrix-based methods to accelerate the similarity search computation and guarantee the accuracy of SimRank score.

2.1.2.2 Matrix-Based Methods

There is a growing body of research on matrix-based methods for SimRank computation (Fujiwara, Nakatsuji, Shiokawa, & Onizuka, 2013; C. Li et al., 2010; Yu, Lin, Zhang, & McCann, 2018a). Li et al. (C. Li et al., 2010) proposed a fast matrix-based similarity search algorithm based on graph topology. This algorithm is motivated by two powerful matrix operators: Kronecker product (Gravano, Garcia-Molina, & Tomasic, 1994), and the well-known Sylvester Equation (Cho, Garcia-Molina, & Page, 1998). Based on the two matrix operators, they re-write the SimRank equation in a non-iterative matrix-based manner. Another improvement of this algorithm is that it uses eigen-value decomposition to reduce memory cost. However, when the target ranking of the low-rank approximation is $n \ll N$, the memory space requirement of Li et al.'s algorithm (C. Li et al., 2010) is $O(N^2n^2)$, which is even higher than the original SimRank algorithm. To bridge this gap, Fujiwara et al. (Fujiwara et al., 2013) proposed a non-iterative algorithm called SimMat. SimMat algorithm also resorts to the Sylvester equation for computing similarity scores. Then, to enhance time and space efficiency, SimMat algorithm avoids unnecessary similarity computations by using Cauchy-Schwarz inequality. The time and space complexities of SimMat reduce to $O(Nn)$ (where $n \ll N$).

2.2 Similarity Search over Evolving Graphs

In real-life applications, while the scale of network structured data is increasing, evolving update often exists in network data. To illustrate, in a typical social network: Facebook, the number of users as well as the relationship between any two nodes change in every second. If we continue to compute similarity scores by static algorithms, all the similarity scores need to be computed again even when there is a slight update in a graph, which is time and space consuming. With regard to the research direction of similarity detection on the graph structure, many scholars have conducted studies on similarity detection over evolving graphs. According to the use of different methods for achieving the similarity detection on dynamic graphs, the existing research can be divided into two: PageRank-based and SimRank-based.

2.2.1 PageRank on Evolving Graphs

PageRank (Brin & Page, 1998) is a fundamental algorithm over evolving graphs. It is also applied extensively to compute nodes significant in a graph. In this algorithm, if two pages get a link, the weight of the page will add one. According to this law, each web page will have a PageRank score. Subsequently, when users search a single word on Google, the results will be ranked by PageRank score of each web page. Then, personalized PageRank (Brin & Page, 1998) was proposed to improve the PageRank algorithm. The salient feature of PPR is that each node will have a node set referred to as seed nodes. When we want to search closer nodes of the target node, we would not need to search through all nodes in the graph. We will obtain the end node from the seed nodes with probabilistic.

As the research on similarity detection on dynamic graphs becomes increasingly popular, dynamic models based on PageRank and personalized PageRank have recently been proposed. (Desikan, Pathak, Srivastava, & Kumar, 2005) put forth a general PageRank algorithm that incrementally computes any metric which satisfies the first order Markov property. (Berberich, Bedathur, Weikum, & Vazirgiannis, 2007) offered a computationally efficient normalisation for PageRank scores that allow their comparison across networks. Berberich et al. further show that, unlike the classic PageRank measure, the normalised PageRank scores are resistant to non-local changes in the graph. (Bahmani, Kumar, Mahdian, & Upfal, 2012) proposed a PageRank algorithm over evolving graphs, wherein the underlying changes to the network are not notified. Mahdian et al. developed two algorithms. The first one is randomized algorithm, based on the idea that nodes with higher PageRank values should be probed more frequently since they have a greater impact on the PageRank values of other nodes. The second one is the deterministic algorithm. Both algorithms achieve a verifiable performance guarantee that outperforms the naive approach, which then traverses the nodes in a round-robin manner.

2.2.2 SimRank on Evolving Graphs

Recall the work proposed by (C. Li et al., 2010), which is briefly introduced in the Section 2.1.2.2. Li et al. not only presented a matrix-base format for the SimRank method but also developed a similarity computation algorithm for dynamic, time-evolving graphs based on the matrix-base format SimRank algorithm. The adjacency matrix changes over time in a dynamic graphs. The incremental SimRank algorithm was the first one to demonstrate an SVD(singular value decomposition) approach for incrementally updating similarities of all node-pairs. Its time complexity is $O(r^4n^2)$, where r denotes the target rank of the low-rank approximation and r is smaller than the node number of the graph. Li et al. also developed two techniques, S_Track and P_Track, for node similarity tracking and node centrality tracking over evolving graphs based on the incremental SimRank algorithm. However, it is possible to improve the accuracy of the incremental SimRank algorithm, given that the incremental algorithm may miss some information. The computation of the incremental algorithm is quartic with response to the value of r. In real applications, the value of r is not much smaller than the node number of the graph; thus, the incremental algorithm is time-consuming.

To bridge these research gaps, Yu et al. proposed a fast incremental SimRank on link-evolving graphs (Yu, Lin, & Zhang, 2014a). The fast incremental SimRank algorithm(Inc-uSR) generates the different similarity scores of each updated link by a rank-one Sylvester equation, instead of incrementally updating SVD (C. Li et al., 2010), and the time complexity of Inc-uSR algorithm is $O(Kn^2)$, where K refers to the iteration times, and n denotes the node number of graphs. To ensure a more efficient incremental SimRank algorithm over evolving graphs, Yu et al. also developed a pruning strategy that is capable of extracting the "unaffected areas" without sacrificing accuracy.

Both incremental SimRank algorithms (C. Li et al., 2010; Yu, Lin, & Zhang, 2014a) focus on all node-pairs similarity search on evolving graphs, which will limit the scalability of the algorithms. Inspired by this idea, Shao et al. developed two-stage random-walk sampling framework(TSF) for SimRank algorithm over large-scale dynamic graphs (Shao, Cui, Chen, Liu, & Xie, 2015a). The first stage (pre-processing stage) of the TSF algorithm utilizes a novel sampling method to index raw random walks from a set of one-way graphs. The indexed walks approach can effectively reduce the redundant sampling. The time complexity of the pre-processing stage is $O(NR_g)$, where N refers to the node number of the graph, and R_g is the number of one-way graphs. Under the second stage (query stage) of TSF algorithm, it can quickly find related vertices by removing unqualified vertices based on one-way graph connectedness. When the error bound is $1 - \epsilon$, TSF can also estimate the SimRank score with a certain probability using additional R_q samples. However, SimRank algorithm suffers from some limitations(Section 4.1.1), such as: “zero issue”, computation efficiency issues, and accuracy issues. The limitations of the SimRank algorithm will be brought to the dynamic SimRank algorithms. To bridge this research gap, we propose a fast and accurate CoSimRank-base similarity search algorithm over dynamic graphs.

2.3 Role-based Network Analysis

Role similarity detection in graph structure is also a widely used graph analysis method. Unlike the similarity detection algorithm we introduced before, role similarity detection can reflect the structural similarity between point pairs and reflect role equivalence. Generally speaking, role equivalence can be divided into four categories: structural equivalence, automorphic equivalence, equitable partition, and regular equivalence (Lee, 2012).

(Batagelj, Doreian, & Ferligoj, 1992) proposed an optimization approach to regular equivalence. The algorithm calculates for how far a structure deviates from a perfectly regular partition. The technique subsequently uses a local optimization procedure to identify partitions that minimised the criterion function. (Borgatti & Everett, 1993) introduced two algorithms for computing regular equivalence. Quantitative data is handled by the REGE algorithm, whereas categorical data is handled by the CATREGE algorithm. Moreover, the CATREGE algorithm is more efficient and accurate than the REGE algorithm.

(Rothe & Schütze, 2014) proposed a ground-breaking study on role analysis over networks, called RoleSim. RoleSim algorithm retrieves role similarity search scores of node-pair by calculating the maximum matching values of in-neighbour matrix. The basic philosophy of the algorithm is that two objects are similar if they are related to similar objects. Accordingly, some role similarity algorithms have been proposed. (Shao et al., 2019) developed an efficient de-anonymization of social networks with structural information called RoleSim++. Different from RoleSim algorithm, RoleSim++ captures information from both in-neighbours as well as out-neighbours. The most state-of-art role similarity algorithm is StructSim (Chen, Lai, Qin, & Lin, 2021). StructSim algorithm makes use of BinCount matching instead of maximum matching. As a result, the algorithm improves the computation efficiency while ensuring admissibility. More details have been introduced in Section 5.7. However, the existing role similarity search algorithm suffers from some limitations in terms of accuracy, computation efficiency, and dynamic. To bridge these research gaps, we propose a more accurate and efficient role similarity search algorithm in this thesis.

3 Fast and Accurate Similarity Search For Evolving Graphs

This chapter discusses the development of an innovative similarity search algorithm for dynamic topology structures. A dynamic graph can be described as a graph updated by an ordered list (*e.g.,* additions and deletions of nodes and edges). In other words, each update in a dynamic graph is known and unique.

Consider Facebook, a typical example of a social network (graph), to comprehend the main idea of a dynamic network. When a person creates a Facebook account, a new node would be created in the social network (`node addition`). Moreover, when they add a friend on Facebook, a "follow" edge is created in the social network (`edge addition`).

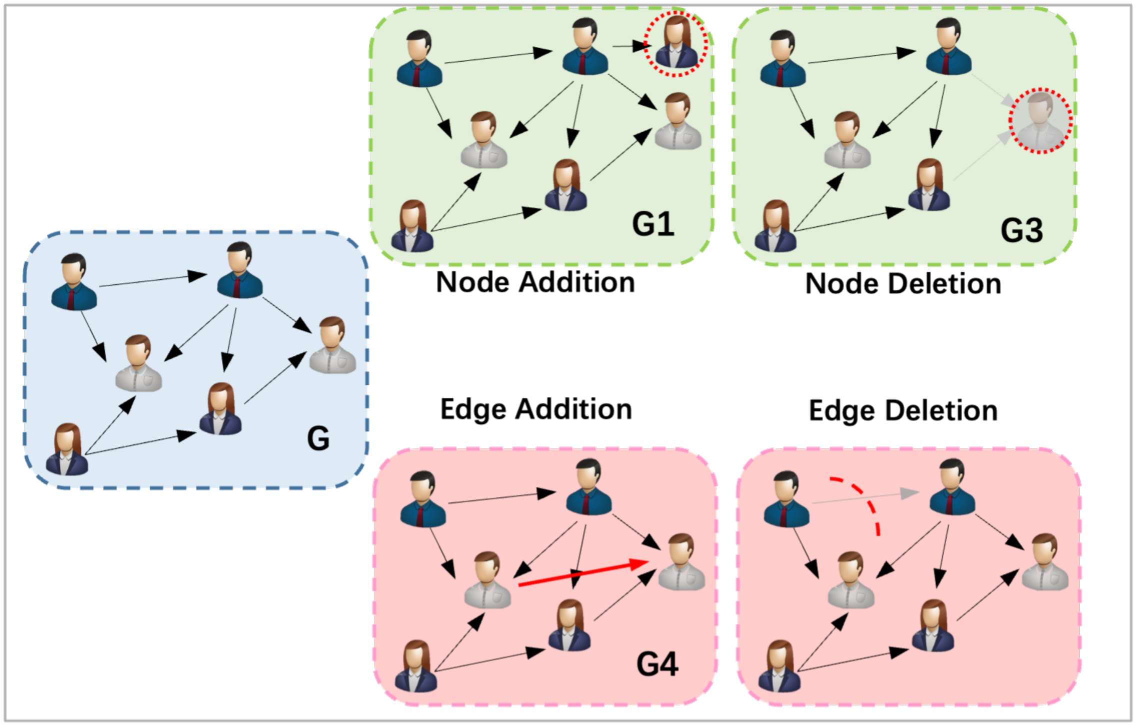


Figure 3.1: Dynamic Graph

In Figure 3.1, G is a small Facebook social network. $(G1, G2)$ describes all possible cases of nodes' update, and $(G3, G4)$ indicates possible cases of edges' update in this dynamic graph.

As shown in Chapter 2, many existing works focus on the similarity search for static graphs or small- or medium-sized dynamic graphs. This chapter presents a novel similarity search algorithm for large-scale dynamic graphs.

3.1 Introduction

Graphs are widely used to model complex objects (*e.g.,* web pages) and their relationships (*e.g.,* hyperlinks). Quantifying objects' similarity based on graph structure topology is a foundational operation for a lot of network research. Examples include web mining (Kinne & Lenz, 2021), graph clustering (Smirnov & Warnow, 2021), Amazon recommendation system (J. Wu et al., 2021), protein structure analysis (Biehn & Lindert, 2021), etc.

Most existing research on similarity search focusing on static graphs have been introduced in Chapter 2. However, in reality, most information systems keep changing constantly. Coping with continuous dynamic changes is the surest way to keep pace with the Information Age and guarantee success with dynamic updates. The challenges associated with these changes are real and highly uncertain. Similarity search in a large-scale dynamic network (graph) inherently presents computation challenges, and the constant updating of data exacerbates the challenges. This is particularly valid for those parts of societies dependent on differences and are sensitive to information changes.

Following are some real-world applications of similarity search, highlighting the need for efficient algorithms to handle such assessments across large-scale dynamic graphs.

Application 1 (Synonym Expansion).

Synonym expansion is a useful tool in search engine query rewriting (Antonellis, Garcia-Molina, & Chang, 2008; Chaudhuri, Ganti, & Xin, 2009) and text simplification (Buyukkokten, Garcia-Molina, & Paepcke, 2001). It replaces a target word in a sentence with another, more appropriate word. The current CoSimRank measure was utilised to compute the similarity between words based on the intuition that "two words that are synonyms of each other should have similar lexical neighbour". In this application, the graph is constructed using nouns, adjectives, and verbs appearing in Wikipedia and form nodes in the graph ($a \in \mathbf{V}$); the rest of the graph represent types of syntactic configurations extracted from the parsed Wikipedia pages (*e.g.,* adjective-noun, verb-object, and noun-noun coordination), which constitute the edges of the graph ($< a \rightarrow b > \in \mathbf{E}$). They evaluated the CoSimRank similarities between words (synonyms). The results from using CoSimRank are superior to identifying effective synonyms using two personalised PageRank vectors (Rothe & Schütze, 2014).

As people's language habits continue changing, the synonyms and phrases in each vocabulary are constantly updated. The current CoSimRank algorithm can efficiently solve the similarity search on the static graph. However, when the static graph is changed to a dynamic one, the algorithm must recompute the similarities of all node pairs with respect to every update, making the current CoSimRank algorithm impractical to use (Rothe & Schütze, 2014).

Application 2 (Lexicon Extraction).

Automatically building bilingual lexicons from corpora is an essential task in natural language processing. Rothe and Schütze (Rothe & Schütze, 2014) applied CoSimRank to lexicon extraction and represented an English and a German text corpus as two graphs, wherein nodes represent words and edges denote the grammatical relationships between the words. The central intuition of CSR is that "a node in the English graph and a node in the German graph are similar (*i.e.,* they are likely to be translations of each other) if their neighbouring nodes are similar". Rothe and Schutze initialised the CoSimRank scores using an English-German "seed" dictionary whose entries correspond to known pairs of equivalent nodes (words). Their approach produced more reliable similarity results than the previous SimRank-based approaches (Laws et al., 2010; Tamura, Watanabe, & Sumita, 2012).

However, with the continuous change of people's language habits, synonyms set for each word and the English-German "seed" dictionary are updated from time to time. Therefore, using the original CoSimRank algorithm to follow the update results in high computation pressure.

It can be noted that CoSimRank is a robust similarity measure between two objects based on static graph topologies. It recursively follows the SimRank-like philosophy that "two nodes are considered similar if their in-neighbours are similar". CoSimRank is a node pair similarity measure, different from PageRank that ranks *nodes* only. Intuitively, a CoSimRank score $s(a, b)$ between nodes a and b aggregates all meeting times of two random surfers starting at a and b, in contrast with SimRank (Jeh & Widom, 2002) that counts their first meeting time only. Thus, CoSimRank has been shown (Rothe & Schütze, 2014) to be more accurate and practical than SimRank in many applications (*e.g.,* synonym expansion and lexicon extraction). However, some major limitations of the existing CoSimRank searching algorithm prevent it from being effectively applied to some real applications, which are illustrated in the section 3.1.1. The detailed definition and current evaluation of SimRank and CoSimRank are reviewed in Chapter 3.2.

3.1.1 Motivation

Previous research on CoSimRank similarity search suffers a few major limitations, of which the lack of support for dynamic graphs is the most important one. The state-of-the-art algorithms for CoSimRank search (Rothe & Schütze, 2014) are mainly restricted to static graphs, where a CoSimRank score is retrieved from the sum of the dot product of two personalised PageRank vectors. However, in many real applications, graphs (*e.g.,* for Facebook and Twitter) are often updated dynamically, with new nodes/edges forming over time. However, it is difficult for the existing static approaches to produce quick responses with evolving graphs due to their cost-inhibitive overheads for recomputing CoSimRank scores from scratch.

This highlights the urgent need to investigate the problem concerning fast and accurate dynamic CoSimRank search. This problem can be formally defined as follows.

Problem (Dynamic CoSimRank on Evolving Graphs).

Given: a graph G, a collection of edge updates ΔG to G, and a query set $Q = \{q_1, q_2, \cdots\}$, here $q_i (q_i \in V)$ is the target node(query).

Retrieve: the changes to the CoSimRank scores with respect to Q on $(G \oplus \Delta G)$ quickly and accurately.

This thesis proposes a fast, accurate dynamic scheme for CoSimRank search over evolving graphs to address this issue. In the remainder of this chapter, the schema will be discussed for two different situations, i.e, D-CoSim (for node and edge additions) and D-deCoSim(for node and edge deletions).

Moreover, as an important application of D-CoSim and D-deCoSim, we show that our dynamic D-CoSim algorithm can also be applied to static graphs to greatly speed up large-scale CoSimRank search. The details of this application are presented in Chapter 4.

3.1.2 Chapter Outlines

This chapter is organised as follows:

Chapter 3.2 In this section, we recall the current SimRank and CoSimRank algorithms and analyse the limitations of both algorithms while applying them for dynamic graphs.

Chapter 3.3.1 An efficient similarity search algorithm D-CoSim for incremental dynamic graphs is proposed in this section.

Chapter 3.3.1.1 A cost-efficient operation of the update adjacency matrix of incremental dynamic graphs in response to each update is detailed in this subsection.

Chapter 3.3.1.2 A novel similarity search algorithm D-CoSim over large-scale incremental dynamic graph is suggested. D-CoSim algorithm can accurately detect the "influenced part" with respect to the updated part of dynamic graphs with high computation efficiency. The time complexity of the innovated similarity search operation is $O(K(\tilde{m} + \tilde{n}p|Q|))$, and it requires $O(\tilde{m} + K\tilde{n})$ memory to compute $\mathbf{\Delta S}[:, Q]$ dynamically after K iterations, where $|Q|$ is the number of queries. In contrast, the current CoSimRank algorithm requires $O(K(m + n))$ time and $O(m + Kn)$ memory to compute the same graph.

Chapter 3.3.2 In this section, we proposed D-deCoSim algorithm to retrieve the similarity scores for decremental dynamic graphs. This algorithm considers several cases with respect to edges' and nodes' deletion.
Thus, we separate D-deCoSim to D-deCoSim(Edges) and D-deCoSim(Nodes) algorithm. D-deCoSim(Edges) is an algorithm to obtain CoSimRank scores of the decremental dynamic graphs with edge deletion. It can compute the CoSimRank scores of update parts only and avoids repeated computation. For decremental dynamic graphs with node deletion, for each deleted node, we separate out-neighbours of a deleted node into three cases based on the number of $\mathcal{D}_d$ and the indegree of each node of $\mathcal{D}_d$, here $\mathcal{D}_d$ is the in-neighbour nodes set of the deleted node. Then compute the CoSimRank scores of "refreshed area" only using the formula corresponding to the three cases respectively in response to the query.

Chapter 3.3.3 In this section, a combination algorithm has been proposed, Com-D, which is composed by D-CoSim, D-deCoSim(Edge) and D-deCoSim(Node). Com-D can efficiently compute similarity values on dynamic graphs.

Chapter 3.4 We conduct many experiments on real datasets to demonstrate that our algorithm D-CoSim and D-deCoSim steadily outperform two state-of-the-art CoSimRank competitors (CSR (Rothe & Schütze, 2014) and CSM (Yu & McCann, 2015a)) with 3-5 order-of-magnitude(time efficiency). Compared with CoSimRank and CSM algorithms, our algorithms show a great advantage in memory efficiency. In particular, CSM cannot be implemented for large graphs since its memory complexity is $O(n^2)$. The experimental results also show that our algorithms do not compromise on accuracy while increasing the (search) speed.

3.2 Preliminaries

In this section, we first define the main notations and symbols used in the remainder of the thesis, and the rest definition of symbols can be found in the List of Symbols.

Definition 1 (**Directed Graph**). *A directed graph (or digraph) is an order pair $G = (V, E)$ where*

- *V is a set of elements, called vertices or nodes;*
- *E is a set of directed edges, and each directed edge is an ordered node pair.*

$|V|$ indicates the cardinality of edges in graph G, and $|E|$ is the number of nodes in G.

Definition 2 (**Dynamic Graph**). *A dynamic graph is a new graph evolving from an old one by applying a sequence of updates. It can be expressed by the mathematical formula: $G = G + \Delta G$. Here. ΔG represents one of the four cases, i.e., the addition of nodes or edges, or the deletion of nodes or edges of graph G.*

Definition 3 (**Vertices Neighbours.**). *Given a directed graph $G = (V, E)$, for any edge $(u \to v) \in E$ ($(u, v) \in V$), the node u is the in-neighbour of node v. At the same time, the node v is the out-neighbour of node u.*

Then, we denoted the following:

- *$\mathcal{I}(v)$ denotes the in-neighbour set of node v, which means all nodes in that set have a direct link to v:*

$$I(v) = \{u \in V | (u \to v) \in E\}$$

- *$\mathcal{O}(v)$ represents the out-neighbour set of node v, which means v has directs links to nodes in that particular set:*

$$O(v) = \{u \in V | (v \to u) \in E\}$$

- *deg_v^- is the number of in-neighbours of node v, i.e., $deg_v^- = |I(v)|$.*

- *deg_v^+ is the number of out-neighbours of node v, i.e., $deg_v^+ = |O(v)|$.*

As presented previously in Chapter 2, SimRank has been proposed as an essential technique in existing works on similarity search. Its basic premise is that "two objects are similar if they are referenced by similar objects". Based on this intuition, there are two models from the SimRank family, namely Jeh and Widom's SimRank model (Jeh & Widom, 2002), and CoSimRank model (Rothe & Schütze, 2014). Their formal definitions and calculation details are provided as follows.

3.2.1 Jeh and Widom's SimRank Model

Following the basic intuition of SimRank, the recursive equation of SimRank is given in Definition 4.

Definition 4 (**SimRank**). *Given a directed graph $G(E, V)$, the similarity score of node pair $(u, v) \in V$ can be retrieved by,*

$$s(u,v) = \begin{cases} 1, & u = v; \\ \dfrac{C}{\deg_u^- \deg_v^-} \displaystyle\sum_{j \in I(v)} \sum_{i \in I(u)} s(i,j), & I(u), I(v) \neq \varnothing; \\ 0, & \textit{otherwise}. \end{cases} \tag{3.1}$$

where $C \in (0,1)$ is the decay factor.

Mathematically, the matrix version of SimRank is formulated as follows:

$$\mathbf{S} = \max\left\{C\mathbf{A}^T\mathbf{S}\mathbf{A}, \mathbf{I}\right\} \tag{3.2}$$

where $\mathbf{S}$ is the SimRank matrix, whose element $S[i,j]$ denotes the similarity score between nodes i and j in a graph G; C is a constant decay factor between 0 and 1, $\mathbf{A}$ is the column-normalised adjacency matrix, and $\mathbf{I}$ is the identity matrix. It can be seen from Eq.(3.2) that the SimRank algorithm computes the first meeting time of two random surfers from nodes i and j in graph G. For each node, the similarity score between the node and itself $(s(a,a), a \in \mathbf{V})$ strictly equals 1.

3.2.2 CoSimRank Model

The second model is the CoSimRank similarity search algorithm proposed by (Rothe & Schütze, 2014). By default, CoSimRank is defined on a digraph. If a graph is undirected, it can be converted into a digraph by replacing every undirected edge between node pair (u,v) with two directed edges $(u \to v)$ and $(v \to u)$. CoSimRank is an feasible node-pair similarity measure based on graph topologies. It is based on the recursive philosophy that "two nodes are considered similar if their in-neighbours are similar". Unlike SimRank (Jeh & Widom, 2002), the CoSimRank score of each node with itself $S[a,a] (a \in \mathbf{V})$ is not constantly 1. Mathematically, CoSimRank is formulated as follows:[1]

$$\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I} \tag{3.3}$$

where $\mathbf{S}$ is the CoSimRank matrix, whose element $\mathbf{S}[i,j]$ is the similarity score between node i and node j in the graph G; C is a constant decay factor between 0 and 1; $\mathbf{A}$ is the column-normalised adjacency matrix; $\mathbf{I}$ is the identity matrix; and $(*)^T$ is the matrix transpose. To normalise all CoSimRank scores into $[0,1]$, one can readily use $S' = (1-C)\mathbf{S}$.

To evaluate one single pair CoSimRank score, Rothe and Schütze (Rothe & Schütze, 2014) adopted a novel method to compute $\mathbf{S}[i,j]$:

$$S[i,j] = \sum_{k=0}^{\infty} C^k (\mathbf{p}_i^{(k)})^T \mathbf{p}_j^{(k)} \tag{3.4}$$

where $\mathbf{p}_j^{(k)}$ is the personalised PageRank vector with respect to the seed node j, which can be iteratively obtained from

$$\mathbf{p}_j^{(k)} = \mathbf{A}\mathbf{p}_j^{(k-1)} \quad \text{with} \quad \mathbf{p}_j^{(0)} = \mathbf{e}_j \tag{3.5}$$

It requires $O(K(m+n))$ time and $O(m+Kn)$ memory to compute a single pair score $\mathbf{S}[i,j]$ using Eqs.(3.4) and (3.5) for the static graph G with n nodes and m edges after K iterations. When

[1] In comparison, SimRank (Jeh & Widom, 2002) is defined as: $\mathbf{S} = \max\{C\mathbf{A}^T\mathbf{S}\mathbf{A}, \mathbf{I}\}$.

the graph is dynamically updated, it will be expensive to recompute all CoSimRank scores from scratch.

We first present our efficient dynamic scheme, D-CoSim, that can quickly and accurately obtain CoSimRank scores for incremental evolving graph streams. Next, D-deCoSim similarity search algorithm for decremental dynamic graphs is illustrated in detail.

3.3 Similarity Search for Large-scale Evolving Graph Streams

In this section, we leverage two efficient and accurate similarity search algorithms for dynamic graphs, namely D-CoSim and D-deCoSim algorithm. We introduce D-CoSim algorithm firstly in subsection 3.3.1.

3.3.1 The Similarity Search algorithm For Incremental Dynamic Graphs

Changes in the graph will directly affect the structure and values of its adjacency matrix A. We propose an efficient measure to update the adjacency matrix with respect to each update of the incremental dynamic graph.

3.3.1.1 Updating Adjacency Matrix

The adjacency matrix needs to be renewed at once with respect to each update of the dynamic graph. To improve the efficiency of updating the adjacency matrix, we chunk all update edges with the same end node and then update the adjacency matrix column to column. There are two reasons to chunk update edges by this method. The first one is that updating the edge set with the same end node, which only affects the end node's column of the adjacency matrix. Secondly, the pre-processing of the adjacency matrix in our algorithm follows column normalisation. The method of updating the adjacency matrix is shown as follows: Given an old graph G and a set of new edges updated to G:

$$\Delta G = \{(v_1 \to u_1), (v_2 \to u_2), (v_3 \to u_3), \cdots\}$$

according to the endpoint u_i of each edge $(v_i \to u_i)$ in ΔG, we first bunch all edges in ΔG into pieces:

$$\Delta G = \Delta G_{u_1} \cup \Delta G_{u_2} \cup \cdots \cup \Delta G_{u_p}$$

such that all edges in each piece ΔG_{u_i} share a common endpoint u_i. Thus, each piece ΔG_{u_i} takes the following form:

$$\Delta G_{u_i} = \{(v_{i_1} \to u_i), (v_{i_2} \to u_i), \cdots, (v_{i_\delta} \to u_i)\}$$

which is expressed as $\Delta G_{u_i} \triangleq ([v_{i_1}, v_{i_2}, \cdots, v_{i_\delta}] \to u_i)$.

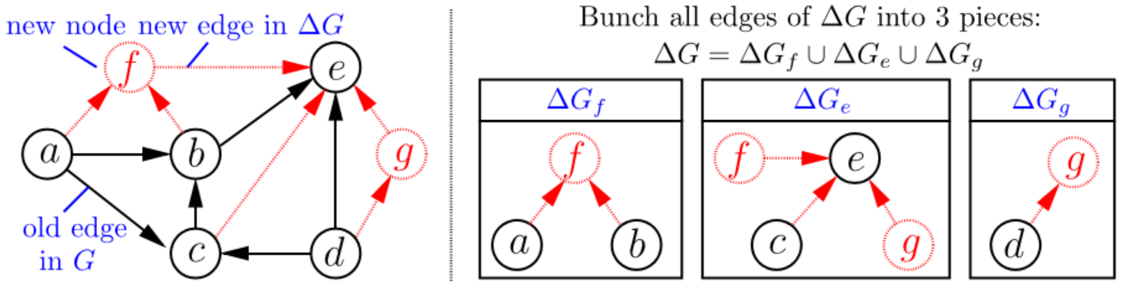


Figure 3.2: Decomposing G into a spanning polytree T and ΔG $(= G \ominus T)$

Example 1. *Figure 3.2 depicts an old graph G (solid black arrows and solid black cycle), and an updated graph ΔG (dashed red arrows and dashed read cycle) to G:*

$$\Delta G = \{(a \to f), (c \to e), (d \to g), (f \to e), (b \to f), (g \to e)\}.$$

To simplify the process of chunking, the partition of the new nodes is determined by the new edges. We lump the edges of ΔG into three pieces: $\Delta G = \Delta G_e \cup \Delta G_f \cup \Delta G_g$, where

$$\Delta G_e = \{(c \to e), (f \to e), (g \to e)\} \triangleq ([c, f, g] \to e),$$

$$\Delta G_f = \{(a \to f), (b \to f)\} \triangleq ([a, b] \to f), \textit{and} \quad \Delta G_g = ([d] \to g). \; \square$$

Remark 1. *If graph G is undirected, we first convert it into a directed graph by replacing each undirected edge between node pair (v, u) with two directed edges $(v \to u)$ and $(u \to v)$. Then, when bunching the edges of ΔG into pieces, we put $(v \to u)$ to ΔG_u, and $(u \to v)$ to ΔG_v.*

Remark 2. *A new node of the old graph G can be defined as follows: for a new edge, one of the two endpoints does not belong to the old graph G. Mathematically, $\{(a \to b) \notin E | a \in V, b \notin V\}$, then edge $(a \to b)$ is a new edge of the old graph G, and node b is a new node. Recall example 1, $\{((a \to f), (b \to f) and (f \to e)) \notin E | (a, b, e) \in V, f \notin V\}$, thus $((a \to f), (b \to f) and (f \to e))$ are three new edges of the old graph G, and node f is a new node of the old graph.*

The way we chunk the edges of ΔG has two advantages. First, we can efficiently characterise the changes to the adjacency matrix $\mathbf{A}$ of an old graph G in response to ΔG_{u_i} as a linear transformation of the u_i-th column of the old $\mathbf{A}$. This characterisation allows us to dynamically capture only the "refreshed areas" of CoSimRank scores in response to the update ΔG_{u_i}, instead employing the original static approach that has to recompute all similarities throughout $G \oplus \Delta G_{u_i}$ from scratch. Second, bunching the edges of ΔG facilitate sharing and reusing common information among all the edge updates over each piece ΔG_{u_i}, thus discarding many unnecessary repeated computations on graph streams. For instance, to efficiently update CoSimRank similarities in response to each piece $\Delta G_{u_i} \triangleq ([v_{i_1}, v_{i_2}, \cdots, v_{i_\delta}] \to u_i)$, the intermediate results to update the edge $(v_{i_1} \to u_i)$, once computed, can be maximally reused to update all other edges (*e.g.,* $(v_{i_2} \to u_i), (v_{i_3} \to u_i), \cdots$) in ΔG_{u_i}. Therefore, D-CoSim is highly efficient for evolving graph streams.

Having bunched together all edges of ΔG into chunks, we propose an efficient approach that dynamically computes the changes to the CoSimRank scores in response to each update piece ΔG_u[2]. We observe that each update piece ΔG_u changes only one column of $\mathbf{A}$. Specifically, we show the following lemma.

Lemma 1. *Given an old graph G and an update piece to G: $\Delta G_u = ([v_1, v_2, \cdots, v_{\delta_u}] \to u)$, the new column-normalised adjacency matrix $\tilde{\mathbf{A}}$ of graph $(G \oplus \Delta G_u)$ can be dynamically updated from old $\mathbf{A}$ by replacing its u-th column with*

$$\tilde{\mathbf{A}}[:, u] = \tfrac{1}{\delta_u + \deg_u^-} \left(\deg_u^- \mathbf{A}[:, u] + \mathbf{1}_{\{v_1, v_2, \cdots, v_{\delta_u}\}} \right) \tag{3.6}$$

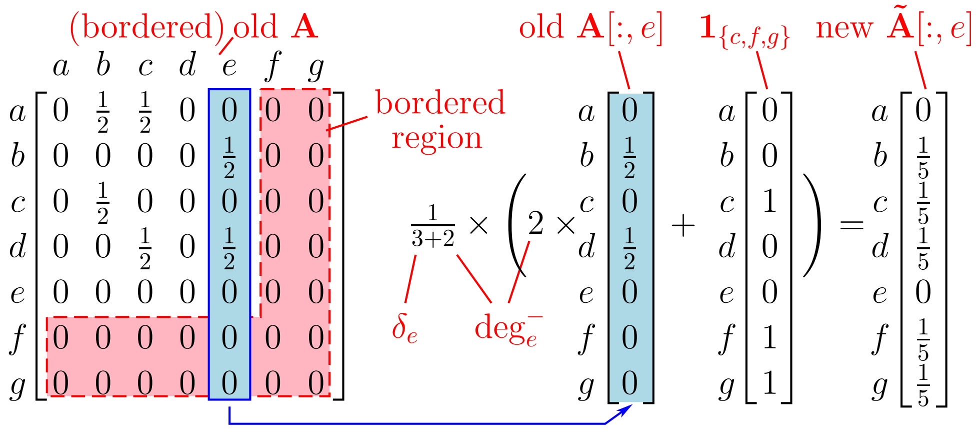
where $\deg_u^-$ is the in-degree of node u in the old graph G; δ_u is the number of edge updates in ΔG_u; and $\mathbf{1}_{\{v_1, v_2, \cdots, v_{\delta_u}\}}$ is a column vector (whose length is the number of rows in new $\tilde{\mathbf{A}}$) with 1s in the $(v_1, v_2, \cdots, v_{\delta_u})$-th entries, and 0s elsewhere.

Note that if the new $\tilde{\mathbf{A}}$ and old $\mathbf{A}$ are not of the same size (occurring when there are new nodes in ΔG_u), then prior to using Eq.(3.6), we should border $\mathbf{A}$ with new zero-columns on the right and new zero-rows at the bottom to make the old one the same size as new $\tilde{\mathbf{A}}$.

Example 2. *In Figure 3.2, the old graph G has five nodes, so the old $\mathbf{A}$ is of size 5×5. In $\Delta G_e = ([c, f, g] \to e)$, there are two new nodes: f and g. Thus, to update $\mathbf{A}$ in response to*

[2]In the following, $\Delta G_{u_i} = ([v_{i_1}, v_{i_2}, \cdots, v_{i_\delta}] \to u_i)$ is abbreviated to $\Delta G_u = ([v_1, v_2, \cdots, v_{\delta_u}] \to u)$ for simplicity.

ΔG_e, we first widen $\mathbf{A}$ to 7×7 with two zero columns and rows: Then, since $\deg_e^- = 2$ and



$$\begin{array}{c|ccccccc} & a & b & c & d & e & f & g \\ \hline a & 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & \frac{1}{2} & 0 & 0 \\ c & 0 & \frac{1}{2} & 0 & 0 & 0 & 0 & 0 \\ d & 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array} \qquad \frac{1}{3+2} \times \left(2 \times \begin{bmatrix} 0 \\ \frac{1}{2} \\ 0 \\ \frac{1}{2} \\ 0 \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 1 \\ 1 \end{bmatrix} \right) = \begin{bmatrix} 0 \\ \frac{1}{5} \\ \frac{1}{5} \\ \frac{1}{5} \\ 0 \\ \frac{1}{5} \\ \frac{1}{5} \end{bmatrix}$$

Figure 3.3: Updating the Adjacency Matrix of the Incremental Dynamic Graph G

$\delta_e = 3$, in light of Eq.(3.6), the e-th column of $\mathbf{A}$, in response to ΔG_e, is updated to

$$\tilde{\mathbf{A}}[:,e] = \tfrac{1}{3+2} \left(2\mathbf{A}[:,e] + \mathbf{1}_{\{c,f,g\}} \right) = [0, \tfrac{1}{5}, \tfrac{1}{5}, \tfrac{1}{5}, 0, \tfrac{1}{5}, \tfrac{1}{5}]^T. \quad \square$$

The computation details can be found in Figure 3.3.

Leveraging Lemma 1, we next show how to update CoSimRank scores dynamically in response to each piece ΔG_u over incremental dynamic graphs.

3.3.1.2 D-CoSim Algorithm

In this subsection, an efficient and accurate similarity search algorithm D-CoSim for CoSimRank scores' computation over incremental dynamic graphs is illustrated in detail. There are two cases associated with the incremental graph, *e.g.*, node addition and edge addition. For nodes' addition, same as updating the adjacency matrix, the size of the similarity score matrix of the graph should be bordered firstly, and then the bordered parts' scores (nodes' addition) and updated parts' scores (edges' addition) can be retrieved by Theorem 1.

Theorem 1. *Given an old graph G, an update piece to G: $\Delta G_u = ([v_1, \cdots, v_{\delta_u}] \to u)$, and a query node $q \in (G \oplus \Delta G_u)$, the changes $\mathbf{\Delta S}[:,q]$ to CoSimRank scores with respect to q are dynamically computed as*

$$\mathbf{\Delta S}[:,q] = \sum_{k=0}^{\infty} C^k \left(\mathbf{t}^{(k)}[q] \cdot \mathbf{p}^{(k)} + \mathbf{p}^{(k)}[q] \cdot \mathbf{t}^{(k)} \right) \tag{3.7}$$

where $\mathbf{p}^{(k)}[q]$ *and* $\mathbf{t}^{(k)}[q]$ *denote the q-th entry of the vectors* $\mathbf{p}^{(k)}$ *and* $\mathbf{t}^{(k)}$*, respectively, which are iteratively obtained by*

$$\begin{cases} \mathbf{p}^{(0)} = \mathbf{e}_u \\ \\ \mathbf{p}^{(k)} = \tilde{\mathbf{A}}^T \mathbf{p}^{(k-1)} \end{cases} \quad \textit{and} \begin{cases} \mathbf{t}^{(0)} = \frac{C}{2(\delta_u + \deg_u^-)} (\mathbf{A} + \tilde{\mathbf{A}})^T \mathbf{r} \\ \\ \mathbf{t}^{(k)} = \tilde{\mathbf{A}}^T \mathbf{t}^{(k-1)} \end{cases} \tag{3.8}$$

in which $\mathbf{A}$ *and* $\tilde{\mathbf{A}}$ *are the old and new column-normalised adjacency matrix of* G *and* $G \oplus \Delta G_u$*, respectively; and* $\mathbf{r} = \lim_{K\to\infty} \mathbf{r}^{(K)}$*, which can be iteratively derived as*

$$\begin{cases} \mathbf{r}^{(0)} = \mathbf{w}^{(K)} \\ \\ \mathbf{r}^{(k)} = C\mathbf{A}^T \mathbf{r}^{(k-1)} + \mathbf{w}^{(K-k)} \end{cases} \qquad (1 \le k \le K) \tag{3.9}$$

$$\textit{with} \begin{cases} \mathbf{w}^{(0)} = \mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u] \\ \\ \mathbf{w}^{(k)} = \mathbf{A}\mathbf{w}^{(k-1)} \end{cases} \qquad (1 \le k \le K) \tag{3.10}$$

Proof. After ΔG_u is updated to G, by the definition in Eq.(3.3), the new CoSimRank scores $(\mathbf{S} + \mathbf{\Delta S})$ in $G \oplus \Delta G_u$ satisfy

$$\mathbf{S} + \mathbf{\Delta S} = C\tilde{\mathbf{A}}^T \left(\mathbf{S} + \mathbf{\Delta S}\right) \tilde{\mathbf{A}} + \mathbf{I}$$

Rearranging the terms in the above equation yields

$$\mathbf{\Delta S} = C\tilde{\mathbf{A}}^T \mathbf{\Delta S} \tilde{\mathbf{A}} + \mathbf{E} \quad \text{with} \quad \mathbf{E} = C\tilde{\mathbf{A}}^T \mathbf{S} \tilde{\mathbf{A}} + \mathbf{I} - \mathbf{S} \tag{3.11}$$

Let $\mathbf{\Delta A} = \tilde{\mathbf{A}} - \mathbf{A}$. From Eq.(3.6) in Lemma 1, we have

$$\mathbf{\Delta A}[:, u] = \tfrac{1}{\delta_u + \deg_u^-} \left(\mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u] \right) \tag{3.12}$$

To simplify $\mathbf{E}$ in Eq.(3.11), we plug $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{\Delta A}[:, u]\mathbf{e}_u^T$, $\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I}$, and let $\mathbf{f}_u \triangleq \mathbf{S}\mathbf{\Delta A}[:$

, u], which produce

$$\begin{aligned}\mathbf{E} &= C(\mathbf{e}_u(\mathbf{f}_u^T\mathbf{A}) + (\mathbf{A}^T\mathbf{f}_u)\mathbf{e}_u^T + (\boldsymbol{\Delta}\mathbf{A}[:,u]^T\mathbf{f}_u)\mathbf{e}_u\mathbf{e}_u^T) \\ &= \mathbf{e}_u\mathbf{x}^T + \mathbf{x}\mathbf{e}_u^T \quad \text{where} \qquad (3.13) \\ \mathbf{x} &= C\mathbf{A}^T\mathbf{f}_u + \tfrac{C}{2}(\boldsymbol{\Delta}\mathbf{A}[:,u]^T\mathbf{f}_u)\mathbf{e}_u \\ &= C(\mathbf{A}^T + \tfrac{1}{2}\mathbf{e}_u\boldsymbol{\Delta}\mathbf{A}[:,u]^T)\mathbf{f}_u \qquad \{\text{using Eq.(3.13)}\} \\ &= \tfrac{C}{2}(\mathbf{A}^T + \tilde{\mathbf{A}}^T)\mathbf{f}_u \qquad (3.14)\end{aligned}$$

Thus, combining Eqs.(3.11) and (3.12), we obtain

$$\begin{aligned}\boldsymbol{\Delta}\mathbf{S} &= \sum_{k=0}^{\infty} C^k(\tilde{\mathbf{A}}^T)^k\mathbf{E}\tilde{\mathbf{A}}^k \qquad \{\text{using Eq.(3.13)}\} \\ &= \sum_{k=0}^{\infty} C^k\left((\tilde{\mathbf{A}}^T)^k\mathbf{e}_u\mathbf{x}^T\tilde{\mathbf{A}}^k + (\tilde{\mathbf{A}}^T)^k\mathbf{x}\mathbf{e}_u^T\tilde{\mathbf{A}}^k\right) \qquad (3.15)\end{aligned}$$

By direct iteration, it follows from Eqs.(3.8) and (3.10) that

$$\mathbf{p}^{(k)} = (\tilde{\mathbf{A}}^T)^k\mathbf{e}_u, \quad \mathbf{w}^{(k)} = \mathbf{A}^k(\mathbf{1}_{\{v_1,\cdots,v_{\delta_u}\}} - \delta_u\mathbf{A}[:,u]) \qquad (3.16)$$

To express $\mathbf{r}$ in $\mathbf{t}^{(0)}$ of Eq.(3.8), by iteration, Eq.(3.9) implies

$$\mathbf{r}^{(K)} = \left(C\mathbf{A}^T\right)^K\mathbf{w}^{(K)} + \left(C\mathbf{A}^T\right)^{K-1}\mathbf{w}^{(K-1)} + \cdots + \mathbf{w}^{(0)}$$

$$= \underbrace{((C\mathbf{A}^T)^K\mathbf{A}^K + (C\mathbf{A}^T)^{K-1}\mathbf{A}^{K-1} + \cdots + \mathbf{I})}_{=\{\text{the first } K \text{ terms of } \mathbf{S} = \sum_{k=0}^{\infty} C^k(\mathbf{A}^T)^k\mathbf{A}^k\}} \underbrace{(\mathbf{1}_{\{v_1,\cdots,v_{\delta_u}\}} - \delta_u\mathbf{A}[:,u])}_{\{\text{By Eq.(3.12)}\}=(\delta_u+\deg_u^-)\boldsymbol{\Delta}\mathbf{A}[:,u]}$$

By applying limits on both sides, we have

$$\mathbf{r} \triangleq \lim_{K\to\infty}\mathbf{r}^{(K)} = \left(\delta_u + \deg_u^-\right)\mathbf{S}\boldsymbol{\Delta}\mathbf{A}[:,u] = \left(\delta_u + \deg_u^-\right)\mathbf{f}_u$$

Thus, by plugging $\mathbf{r} = \left(\delta_u + \deg_u^-\right)\mathbf{f}_u$ into Eq.(3.8), we get

$$\mathbf{t}^{(0)} = \tfrac{C}{2}(\mathbf{A} + \tilde{\mathbf{A}})^T\mathbf{f}_u = \{\text{By Eq.(3.14)}\} = \mathbf{x}, \quad \mathbf{t}^{(k)} = (\tilde{\mathbf{A}}^k)^T\mathbf{x} \qquad (3.17)$$

Substituting Eqs.(3.16) and (3.17) into Eq.(3.15) provides

$$\boldsymbol{\Delta}\mathbf{S} = \sum_{k=0}^{\infty} C^k\left(\mathbf{p}^{(k)}\left(\mathbf{t}^{(k)}\right)^T + \mathbf{t}^{(k)}\left(\mathbf{p}^{(k)}\right)^T\right)$$

Finally, post-multiplying both sides by $\mathbf{e}_q$ yields Eq.(3.7). □

The proof shows the correctness of Theorem 1. Next, let's take an example to see how the D-CoSim algorithm works in a graph.

Example 3. *Recall the old G (solid arrows) in Figure 3.2 and update piece $\Delta G_e = ([c, f, g] \to e)$ to G (dashed arrows). Given query $q = e$, number of iterations $K = 3$, and decay factor $C = 0.6$, Theorem 1 retrieves $\mathbf{\Delta S}[:, e]$ as follows:*

First, we compute $\{\mathbf{w}^{(k)}\}$ and $\{\mathbf{r}^{(k)}\}$ using Eqs.(3.10) and (3.9):

k	$\mathbf{w}^{(k)}$	$\mathbf{r}^{(k)}$
0	$[0, -1.5, 1, -1.5, 0, 1, 1]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$
1	$[-.25, 0, -.75, .5, 0, 0, 0]^T$	$[-.375, 0, 0, -.375, 0, 0, 0]^T$
2	$[-.375, 0, 0, -.375, 0, 0, 0]^T$	$[-.25, -.113, -.975, .5, -.113, 0, 0]^T$
3	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, -1.868, 1.075, -1.5, .116, 1, 1]^T$

Next, we obtain $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ via Eq.(3.8) with $\mathbf{r} = \mathbf{r}^{(3)}$:

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 0, 0, 0, 1, 0, 0]^T$	$[0, .065, -.09, 0, -.105, 0, 0]^T$
1	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, -.045, 0, 0, -.005, 0, 0]^T$
2	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, -.009, 0, 0]^T$
3	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$

Finally, we use Eq.(3.7) to derive $\mathbf{\Delta S}[:, e]$ in response to ΔG_e:

$$\begin{aligned}\mathbf{\Delta S}[:, e] &= \sum_{k=0}^{3} 0.6^k \left(\mathbf{t}^{(k)}[e] \cdot \mathbf{p}^{(k)} + \mathbf{p}^{(k)}[e] \cdot \mathbf{t}^{(k)}\right) \\ &= [0, .0645, -.09, 0, -.2091, 0, 0]^T\end{aligned}$$

□

Theorem 1 implies an efficient dynamic method, D-CoSim, to retrieve the changes to CoSimRank scores. The details of Theorem 1 are demonstrated in Algorithm 1 later.

The algorithm, named D-CoSim, retrieves the changes to CoSimRank scores over the incremental dynamic graph, as depicted in Algorithm 1. Given an old graph G, an update ΔG to G, and a query set Q, it first divides ΔG into pieces $\{\Delta G_u\}$ such that $\Delta G = \bigcup_u \Delta G_u$ with edges in each piece ΔG_u sharing a common end node u (line 2). Then, for each piece ΔG_u, Eq.(3.10) is used to iteratively obtain $\{\mathbf{w}^{(k)}\}$ (line 8 and line 10), by which $\mathbf{r}$ is iteratively derived using

```
Algorithm 1: D-CoSim $(G, \Delta G, C, Q, K)$
  Input  : an old graph $G$, a set of edge updates $\Delta G$ to $G$, decay factor $C$, a query set $Q$, and
           #-iteration $K$
  Output: CoSimRank changes $\mathbf{\Delta S}[:, Q]$ in response to $\Delta G$.
2  chunk all edges of $\Delta G$ to $\{\Delta G_u\}$ s.t. $\Delta G = \bigcup_u \Delta G_u$ and edges in $\Delta G_u = ([v_1, \cdots, v_{\delta_u}] \to u)$
    share common end $u$
4  foreach query $q \in Q$ do initialise $\mathbf{\Delta S}[:, q] := \mathbf{0}$
6  foreach piece $\Delta G_u$ do
8      initialise $\mathbf{w}^{(0)} := \mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u]$
10     for $k = 1$ to $K$ do update $\mathbf{w}^{(k)} := \mathbf{A}\mathbf{w}^{(k-1)}$
12     initialise $\mathbf{r} := \mathbf{w}^{(K)}$
14     for $k = 1$ to $K$ do update $\mathbf{r} := C\mathbf{A}^T\mathbf{r} + \mathbf{w}^{(K-k)}$
16     $\mathbf{\Delta A}[:, u] := \frac{1}{\delta_u + \deg_u^-}\big(\mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u]\big)$
18     update $\tilde{\mathbf{A}} := \mathbf{A} + \mathbf{\Delta A}[:, u]\mathbf{e}_u^T$
20     initialise $\mathbf{p}^{(0)} := \mathbf{e}_u$
22     for $k = 1$ to $K$ do update $\mathbf{p}^{(k)} := \tilde{\mathbf{A}}^T\mathbf{p}^{(k-1)}$
24     initialise $\mathbf{t}^{(0)} := \frac{C}{2(\delta_u + \deg_u^-)}(\mathbf{A} + \tilde{\mathbf{A}})^T\mathbf{r}$
26     for $k = 1$ to $K$ do update $\mathbf{t}^{(k)} := \tilde{\mathbf{A}}\mathbf{t}^{(k-1)}$
28     foreach query $q \in Q$ do
30         initialise $\mathbf{s} := \mathbf{0}$;
32         for $k = 0$ to $K$ do
34             $\mathbf{s} := \mathbf{s} + C^k(\mathbf{t}^{(k)}[q] \cdot \mathbf{p}^{(k)} + \mathbf{p}^{(k)}[q] \cdot \mathbf{t}^{(k)})$;
36         update $\mathbf{\Delta S}[:, q] := \mathbf{\Delta S}[:, q] + \mathbf{s}$;
38     update $\mathbf{A} := \tilde{\mathbf{A}}$
40 return $\mathbf{\Delta S}[:, Q] := \{\mathbf{\Delta S}[:, q] \mid \forall q \in Q\}$;
```

Eq.(3.9) (line 12 and line 14). Next, using $\mathbf{r}$ and Lemma 1, $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ are iteratively computed from Eq.(3.8) (line 20 to line 26). Finally, for each query $q \in Q$, $\mathbf{\Delta S}[:, q]$ is obtained through Eq.(3.7) from the linear combination of the final result $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ (line 32 to line 36).

Correctness. To prove the correctness of D-CoSim, we prove that $\mathbf{\Delta S}[:, Q]$ (line 40) retrieved by D-CoSim is the *correct* CoSimRank change in response to the update ΔG to G. While Theorem 1 guarantees the correctness of $\mathbf{\Delta S}$ with respect to *the one piece* update ΔG_u only, the following theorem further guarantees that, after the one piece update ΔG_u is processed, other pieces being processed will not distort the correct CoSimRank results $\mathbf{\Delta S}[:, Q]$.

Theorem 2. *Let $\Delta G \triangleq \{\Delta G_1, \Delta G_2, \cdots, \Delta G_p\}$ be a set of edges bunched into pieces updated in the old graph G (line 2). The CoSimRank changes $\mathbf{\Delta S}$ (line 40) returned by D-CoSim are correct in response to the updated graph ΔG.*

Proof. Let $\mathbf{\Delta A}, \mathbf{\Delta A_1}, \mathbf{\Delta A_2}, \cdots, \mathbf{\Delta A_p}$ be the changes to the column-normalised adjacency matrices with respect to the graph updates $\Delta G, \Delta G_1, \Delta G_2, \cdots, \Delta G_p$, respectively.

In the 1st round of the for loop (Line 8–38): D-CoSim starts by taking account of G_0 ($\triangleq G$) as the old graph and $\mathbf{S_0}$ ($\triangleq \mathbf{S}$) as the old CoSimRank scores and updates the 1st chunk ΔG_1 to G_0. Theorem 1 ensures that $\mathbf{s}$ (line 36) in the 1st round, denoted by $\mathbf{\Delta S_1}$, is the CoSimRank changes with respect to the update ΔG_1 to G_0, *i.e.,* $\mathbf{\Delta S_1}$ satisfies

$$\mathbf{S_0} + \mathbf{\Delta S_1} = C(\mathbf{A_0} + \mathbf{\Delta A_1})^T(\mathbf{S_0} + \mathbf{\Delta S_1})(\mathbf{A_0} + \mathbf{\Delta A_1}) + \mathbf{I}$$

Then, D-CoSim updates $\mathbf{\Delta S}$ (line 36) by adding $\mathbf{s}$ ($= \mathbf{\Delta S_p}$), updating the current graph from G_0 to G_1 (line 38):

$$\mathbf{\Delta S} = \mathbf{0} + \mathbf{\Delta S_1} = \mathbf{\Delta S_1}, \qquad \mathbf{A_1} = \mathbf{A_0} + \mathbf{\Delta A_1}$$

In the 2nd round of the for loop (Line 8–38): D-CoSim regards $G_1 (= G_0 + \Delta G_1)$ as the old graph and $\mathbf{S_1} (= \mathbf{S_0} + \mathbf{\Delta S_1})$ as the old CoSimRank scores and updates the 2nd chunk ΔG_2 to G_1. Theorem 1 ensures that $\mathbf{s}$ (line 36) in the 2nd round, denoted by $\mathbf{\Delta S_2}$, is the CoSimRank changes with respect to the update ΔG_2 to G_1, *i.e.,* $\mathbf{\Delta S_2}$ satisfies

$$\mathbf{S_1} + \mathbf{\Delta S_2} = C(\mathbf{A_1} + \mathbf{\Delta A_2})^T(\mathbf{S_1} + \mathbf{\Delta S_2})(\mathbf{A_1} + \mathbf{\Delta A_2}) + \mathbf{I}$$

Then, D-CoSim updates $\mathbf{\Delta S}$ (line 36) by adding $\mathbf{s}$ ($= \mathbf{\Delta S_2}$), updating the current graph from

G_1 to G_2 (line 38):

$$\mathbf{\Delta S} = \mathbf{\Delta S_1} + \mathbf{\Delta S_2}, \qquad \mathbf{A_2} = \mathbf{A_1} + \mathbf{\Delta A_2}$$

The for loop (lines 16–38) continues till the last chunk ΔG_p is updated. In the p-th (last) round of for loop (lines 8–36): D-CoSim regards G_{p-1} ($= G_{p-2} + \Delta G_{p-1}$) as the old graph and $\mathbf{S_{p-1}}(= \mathbf{S_{p-2}} + \mathbf{\Delta S_{p-1}})$ as the old CoSimRank scores and updates the p-th chunk ΔG_p to G_{p-1}. Theorem 1 ensures that $\mathbf{s}$ (line 36) in the p-th round, denoted by $\mathbf{\Delta S_p}$, is the CoSimRank changes with respect to the update ΔG_p to G_{p-1}, *i.e.,* $\mathbf{\Delta S_p}$ satisfies

$$\begin{aligned} \mathbf{S_{p-1}} + \mathbf{\Delta S_p} = C(\mathbf{A_{p-1}} + \mathbf{\Delta A_p})^T \cdot (\mathbf{S_{p-1}} + \mathbf{\Delta S_p}) \cdot \\ \cdot (\mathbf{A_{p-1}} + \mathbf{\Delta A_p}) + \mathbf{I} \end{aligned} \tag{3.18}$$

Then, D-CoSim updates $\mathbf{\Delta S}$ (Line 36) by adding $\mathbf{s}$ ($= \mathbf{\Delta S_p}$), updating the current graph from G_{p-1} to G_p (Line 38):

$$\mathbf{\Delta S} = (\mathbf{\Delta S_1} + \cdots + \mathbf{\Delta S_{p-1}}) + \mathbf{\Delta S_p}, \quad \mathbf{A_p} = \mathbf{A_{p-1}} + \mathbf{\Delta A_p}$$

Finally, we check if $\mathbf{\Delta S}$ ($= \mathbf{\Delta S_1} + \mathbf{\Delta S_2} + \cdots + \mathbf{\Delta S_p}$) is the correct CoSimRank change with respect to the update ΔG to G. Our above analysis for each round of the for loop implies

$$\mathbf{S_i} = \mathbf{S_{i-1}} + \mathbf{\Delta S_i}, \qquad \mathbf{A_i} = \mathbf{A_{i-1}} + \mathbf{\Delta A_i} \qquad (\forall i = 1, \cdots, p-1)$$

Repeatedly applying the above iterations provides

$$\begin{aligned} &\mathbf{A_{p-1}} + \mathbf{\Delta A_p} = (\mathbf{A_{p-2}} + \mathbf{\Delta A_{p-1}}) + \mathbf{\Delta A_p} = \cdots = \\ =&(\mathbf{A_0} + \mathbf{\Delta A_1}) + \mathbf{\Delta A_2} + \cdots + \mathbf{\Delta A_{p-1}} + \mathbf{\Delta A_p} \\ =&\mathbf{A_0} + \mathbf{\Delta A} \quad \text{with} \quad \mathbf{\Delta A} \triangleq \mathbf{\Delta A_1} + \cdots + \mathbf{\Delta A_p} \end{aligned} \tag{3.19}$$

Similarly,

$$\mathbf{S_{p-1}} + \mathbf{\Delta S_p} = \mathbf{S_0} + \mathbf{\Delta S} \;\text{ with }\; \mathbf{\Delta S} \triangleq \mathbf{\Delta S_1} + \cdots + \mathbf{\Delta S_p} \tag{3.20}$$

Plugging Eqs.(3.19) and (3.20) into Eq.(3.18) gives

$$\mathbf{S_0} + \mathbf{\Delta S} = C(\mathbf{A_0} + \mathbf{\Delta A})^T(\mathbf{S_0} + \mathbf{\Delta S})(\mathbf{A_0} + \mathbf{\Delta A}) + \mathbf{I}$$

Thus, $\mathbf{\Delta S}$ satisfies the CoSimRank definition, which implies that $\mathbf{\Delta S}$ ($= \mathbf{\Delta S_1} + \mathbf{\Delta S_2} + \cdots + \mathbf{\Delta S_p}$) retrieved by D-CoSim is exactly the CoSimRank changes with respect to the graph update ΔG ($= \Delta G_1 + \cdots + \Delta G_p$) to G_0 ($\triangleq G$). □

We show that $\mathbf{\Delta S}[:, Q]$ returned by D-CoSim (Line 40) is the *correct* CoSimRank change in response to the updated graph ΔG to G. Next, the example presented in Figure 3.2 is used to show how to retrieve the CoSimRank scores of "refreshed are" by Algorithm 1.

Example 4. *Recall the old graph G (solid arrows) and updated graph ΔG to G (dashed arrows) in Figure 3.2. Given the query $q = e$, number of iterations $K = 3$, and decay factor $C = 0.6$, D-CoSim computes $\mathbf{\Delta S}$ in response to ΔG through the following steps.*

First, D-CoSim chunks all edges of ΔG into three pieces: $\Delta G = \Delta G_e \cup \Delta G_f \cup \Delta G_g$, according to Example 1.

Then, the changes of CoSimRank, $\mathbf{\Delta S_1}[:, e]$, with respect to the 1st piece update ΔG_e to G_0 $(= G)$ are derived (see Example 3):

$$\mathbf{\Delta S_1}[:, e] = [0, .0645, -.09, 0, -.2091, 0, 0]^T,$$

Thereafter, D-CoSim considers $G_1(= G_0 \oplus \Delta G_e)$ as the old graph and computes the changes $\mathbf{\Delta S_2}[:, e]$ with respect to the 2nd piece update ΔG_f to G_1:

$$\mathbf{\Delta S_2}[:, e] = [0, .009, 0, 0, .0239, .1, 0]^T.$$

Next, it regards $G_2(= G_1 \oplus \Delta G_g)$ as the old graph and computes the changes $\mathbf{\Delta S_3}[:, e]$ with respect to the 3rd piece update ΔG_g to G_2:

$$\mathbf{\Delta S_3}[:, e] = [0, .018, 0, 0, .0288, 0, .12]^T.$$

Finally, the changes of CoSimRank $\mathbf{\Delta S}[:, e]$ with respect to the graph update ΔG $(= \Delta G_e \oplus \Delta G_f \oplus \Delta G_g)$ are as follows:

$$\begin{aligned}\mathbf{\Delta S}[:, e] &= \mathbf{\Delta S_1}[:, e] + \mathbf{\Delta S_2}[:, e] + \mathbf{\Delta S_3}[:, e] \\ &= [0, .0915, -.09, 0, -.1564, .1, .12]^T \qquad \square\end{aligned}$$

Complexity. We now analyse the computational cost of D-CoSim. Let $\tilde{n}$ and $\tilde{m}$ denote the number of nodes and edges in the new graph $G \oplus \Delta G$, respectively. Let δ be the number of edges in ΔG, and p be the number of update pieces $\{\Delta G_u\}$ in ΔG. Clearly, $p \leq \delta$. The complexity bound of D-CoSim can be defined as the following theorem.

Theorem 3. *D-CoSim requires $O(K(\tilde{m} + \tilde{n}p|Q|))$ time and $O(\tilde{m} + K\tilde{n})$ memory to dynamically calculate $\mathbf{\Delta S}[:, Q]$ after K iterations, where $|Q|$ is the number of queries in Q.*

Proof. D-CoSim runs in three phases: (1) bunching edges of ΔG (Line 2), (2) $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ iteration (Line 8–26), and (3) online query (Line 28–36). Specifically, bunching the edges of ΔG requires $O(\delta)$ time and $O(\delta)$ memory for a linear scan of all edges in ΔG. To iteratively compute $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$, for each query $q \in Q$ and each piece update ΔG_u, it is required $O(K\tilde{m})$ time and $O(\tilde{m} + K\tilde{n})$ memory using Eqs.(3.8), Eqs.(3.9), and Eqs.(3.10). The $O(K\tilde{m})$ time is dominated by five matrix-vector products: $\mathbf{A}\mathbf{w}^{(k-1)}$ (Line 10), $\mathbf{A}^T\mathbf{r}^{(k-1)}$ (Line 14), $\tilde{\mathbf{A}}^T\mathbf{p}^{(k-1)}$ (Line 22), $(\mathbf{A} + \tilde{\mathbf{A}})^T\mathbf{r}$ (Line 24), and $\tilde{\mathbf{A}}\mathbf{t}^{(k-1)}$ (Line 26). The memory $O(\tilde{m} + K\tilde{n})$ is determined by the storage of matrix $\mathbf{A}$, and the resulting iterative vectors. For online query, once $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ are computed, they are saved and reused to compute $\mathbf{\Delta S}[:, q]$ for every query in Q. After $\mathbf{\Delta S}[:, q]$ is updated in response to each piece ΔG_u, all the vectors $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ are freed for the next piece update. Thus, for $|Q|$ queries and p update pieces, D-CoSim algorithm entails $O(K(\tilde{m} + \tilde{n}p|Q|))$ time and $O(\tilde{m} + K\tilde{n})$ memory in total. $\square$

Theorem 3 guarantees the high efficiency of D-CoSim for dynamic CoSimRank search, whose speed is increased by (a) our characterisation of the “refreshed areas” $\mathbf{\Delta S}[:, q]$ in terms of the linear combination of $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ only, and (b) maximally reusing and sharing common intermediate results in response to the edge updates on each piece ΔG_u. In comparison, the existing approach (Rothe & Schütze, 2014) requires $O(K(\tilde{m}+\tilde{n}))$ time to compute only a single-pair $\tilde{\mathbf{S}}[i, j]$ per edge update using Eqs.(3.4) and (3.5) from scratch, leading to $O(K(\tilde{m}+\tilde{n}|Q|)\tilde{n}\delta)$ a total time of to compute $\tilde{\mathbf{S}}[:, Q]$ ($\tilde{n} \times |Q|$ pairs) for δ edge updates, which is rather expensive.

3.3.2 Similarity Search over Decremental Graph

As presented in Figure 3.1, there are four cases of dynamic graph evolution: the addition of nodes, the addition of edges, the deletion of nodes and the deletion of edges. Previously, in Subsection 3.3.1, an efficient similarity search algorithm D-CoSim over incremental dynamic graphs was explained in detail with several examples. However, decremental cases of dynamic graphs also commonly exist. For example, when user signs off their Facebook account, then a node and some related edges need to be removed from the social network (graph). This section describes a novel similarity search algorithm over decremental dynamic graphs. We named this algorithm as D-deCoSim.

In the following part of this section, D-deCoSim is discussed considering two separated cases with respect to the type of updating parts (edges' deletion and nodes' deletion). For better understanding, we define D-deCoSim(Edge) as the CoSimRank similarity search algorithm over decremental dynamic graphs with edge deletion, and D-deCoSim(Node) as the CoSimRank scores' computation algorithm over decremental dynamic graphs with node deletion.

3.3.2.1 Similarity Search over Decremental Dynamic Graphs with Edges' Deletion

This subsection illustrates how the scores can be efficiently computed when some edges are deleted from an old graph. In this case, since the number of nodes in the new graph does not change, the size of the CoSimRank score matrix of the new graph is the same as that of the old graph.

The values and structure of the adjacency matrix closely follow the graph's updating. The adjacency matrix of the old graph $\mathbf{A}$ and the new adjacency matrix of the new graph $\widetilde{\mathbf{A}}$ are the primary information input of the similarity search algorithm. Thus, we introduce the efficient method for updating the adjacency matrix with respect to each update (edge deletion) as follows.

Efficient Updating of the Adjacency Matrix (Edges' Deletion)

Same as D-CoSim presented in the previous subsection, the first step for D-deCoSim in edges' deletion is updating the adjacency matrix. In decremental dynamic graphs, the method of updating the adjacency matrix is different from the one used in D-CoSim, especially when the number of deleted edges with end node u is the same as the in-degree of node u in the old graph, meaning all incoming edges of node u are deleted. Therefore, the entire u^{th} column of the new adjacency matrix consists of zeros. The efficient operation of updating the adjacency matrix concerning dynamic graphs with edge deletion is shown in Lemma 2.

Lemma 2. *Given an old graph G and a deletion update piece of $\Delta G_u = ([v_1, \cdots, v_{\delta_u}] \to u)$, the new column-normalised adjacency matrix $\widetilde{\mathbf{A}}$ of the new graph $\widetilde{G} = G \ominus \Delta G$ can be dynamically updated from the old adjacency matrix A by replacing its u^{th} column:*

$$\widetilde{\mathbf{A}}[:,u] = \begin{cases} \frac{1}{\delta_u - \deg_u^-}(\deg_u^- * \mathbf{A}[:,u] - 1_{\{v_1, v_2, \cdots, v_{\delta_u}\}}) & if \;\; \deg_u^- \neq \delta_u \\ \\ \mathbf{0} & if \deg_u^- = \delta_u \end{cases} \tag{3.21}$$

Here, $\mathbf{0}$ is a one-column matrix, and all its entries are zero.

Regarding decremental dynamic graphs with edge deletion, we assume that no node is deleted from the old graph even if an isolated vertex caused the deletion (the case of node deletion will be discussed in detail in the next subsection). Therefore, the number of nodes of the new graph $\widetilde{G}$ is the same as that of the old graph G. Thus, the size of the adjacency matrix and the CoSimRank scores matrix of the old and new graphs do not change. Next, we use an example to illustrate the updating of the adjacency matrix of a dynamic graph with edge deletion by applying Lemma 2 as follows.

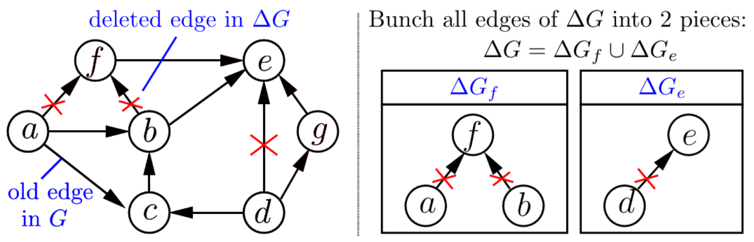

deleted edge in ΔG
old edge in G
Bunch all edges of ΔG into 2 pieces:
$\Delta G = \Delta G_f \cup \Delta G_e$
ΔG_f
ΔG_e

Figure 3.4: Edge Deletion ΔG of Graph G

Example 5. *In Figure 3.4, given an old Graph G, the first deleted pieces is: $\Delta G_e = ([d] \to e)$ and $\Delta G_f = ([a, b] \to f)$.*

Firstly, since $\delta_e = 1$ and $\deg_e^- = 4$, in light of Eq. (3.21), the e^{th} column of $\mathbf{A}$ in response

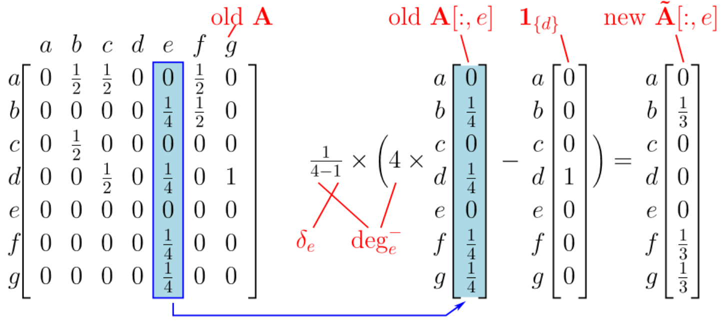


Figure 3.5: Updating the Adjacency Matrix of Dynamic Graph G With Edges Deletion

to ΔG_e is updated to

$$\widetilde{\mathbf{A}}[:,e] = \frac{1}{4-1}(4\mathbf{A}[:,e] - 1_{\{d\}}) = [\,0\ \tfrac{1}{3}\ 0\ 0\ 0\ \tfrac{1}{3}\ \tfrac{1}{3}\,]^T$$

Figure 3.5 shows the computation details concerning the updating of the adjacency matrix.

Then, we consider the next update piece ΔG_f to update the adjacency matrix. The in-degree of node f in the old graph G is the same as δ_f. According to Lemma 2, the f^{th} column of the new adjacency matrix is $\widetilde{\mathbf{A}}[:,f] = [\,0\ 0\ 0\ 0\ 0\ 0\ 0\,]^T$.

D-deCoSim (Edge) over Decremental Dynamic Graphs with Edge Deletion

Leveraging Lemma 2, we next demonstrate how to dynamically update the CoSimRank scores of "refreshed are" in response to a piece of edge deletion. The algorithm is shown in Theorem 4.

Theorem 4. *Given an old graph G, a query $q \in \widetilde{G}(G \ominus \Delta G)$, and a deletion update piece to G: $\Delta G_u = ([v_1, \cdots, v_{\delta_u}] \to u)$, the changes $\Delta \mathbf{S}[:, q]$ to CoSimRank scores with respect to q can be computed by,*

$$\Delta \mathbf{S}[:, q] = \sum\nolimits_{k=0}^{\infty} C^k (\mathbf{p}^{(k)}[q](\mathbf{t}^{(k)}) + \mathbf{t}^{(k)}[q](\mathbf{p}^{(k)})) \tag{3.22}$$

where $\mathbf{p}^{(k)}[q]$ and $\mathbf{t}^{(k)}[q]$ denote the q-th entry of the vectors $\mathbf{p}^{(k)}$ and $\mathbf{t}^{(k)}$, respectively, which are iteratively obtained as follows:

$$\begin{cases} \mathbf{p}^{(0)} = \mathbf{e}_u \\ \mathbf{p}^{(k)} = \tilde{\mathbf{A}}^T \mathbf{p}^{(k-1)} \end{cases} \qquad \begin{cases} \mathbf{t}^{(0)} = \begin{cases} \frac{C}{2(\delta_u - \deg_u^-)} (\mathbf{A} + \tilde{\mathbf{A}})^T \mathbf{r} & if \ \deg_u^- \neq \delta_u \\ \frac{C}{2} \left(\mathbf{A} + \widetilde{\mathbf{A}}\right)^T \mathbf{r}^{(K)} & if \ \deg_u^- = \delta_u \end{cases} \\ \mathbf{t}^{(k)} = \tilde{\mathbf{A}}^T \mathbf{t}^{(k-1)} \end{cases} \tag{3.23}$$

and $\mathbf{r} = \lim_{K \to \infty} \mathbf{r}^{(K)} (1 \leq k \leq K)$, which can be iteratively derived as,

$$\begin{cases} \mathbf{r}^{(0)} = \mathbf{w}^{(K)} \\ \mathbf{r}^{(k)} = C \mathbf{A}^T \mathbf{r}^{(k-1)} + \mathbf{w}^{(K-k)} \end{cases} \qquad \begin{cases} \mathbf{w}^{(0)} = \begin{cases} \delta_u \mathbf{A}[:, u] - \mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} & if \ \deg_u^- \neq \delta_u \\ -\mathbf{A}[:, u] & if \ \deg_u^- = \delta_u \end{cases} \\ \mathbf{w}^{(k)} = \mathbf{A} \mathbf{w}^{(k-1)} \end{cases} \tag{3.24}$$

where $\deg_u^-$ is the in-degree of node u in the old graph, and δ_u is the number of edge updates in ΔG. To ensure that the algorithm is universally applicable, when $\deg_u^- = \delta_u$, $\mathbf{t}^{(0)} = \frac{C}{2} \left(\mathbf{A} + \widetilde{\mathbf{A}}\right)^T \mathbf{r}^{(K)}$ and $\mathbf{w}^{(0)} = -\mathbf{A}[:, u]$.

Next, we use the example in Figure 3.4 to illustrate how Theorem 4 updates the CoSimRank scores of "refreshed area" in response to query q.

Example 6. *Figure 3.4 depicts the old graph G (black arrows) and the updated graph ΔG (black arrows with red cross) to G: $\Delta G_e = ([d] \to e)$ and $\Delta G_f = ([a, b] \to f)$, and we take node c as query, number of iteration $K = 4$, and decay factory is $C = 0.6$. Following Theorem 4, the new similarity score $\Delta\mathbf{S}[:, c]$ can be calculated.*

First, we generate $\Delta\mathbf{S}[:, c]$ with respect to ΔG_e, after the edge $(d \to e)$ has been deleted. Because the in-degree of node e is not equal to 0 in the new graph $G_1 = G \ominus \Delta G_e$, according to Eq.(3.23) and Eq.(3.24), we set $\mathbf{w}^{(0)} = \delta_u \mathbf{A}[:, u] - \mathbf{1}_{\{v_1,\cdots,v_{\delta_u}\}}$ and $\mathbf{t}^{(0)} = \frac{C}{2(\delta_u - \deg_u^-)}(\mathbf{A} + \tilde{\mathbf{A}})^T \mathbf{r}$ to generate $\Delta\mathbf{S}[:, c]$ as follows:

1. *We compute $\mathbf{w}^{(k)}$ and $\mathbf{r}^{(k)}$ using Eq.(3.24):*

k	$\mathbf{w}^{(k)}$	$\mathbf{r}^{(k)}$
0	$[0, 0.25, 0, -0.75, 0, 0.25, 0.25]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$
1	$[0.25, 0.125, 0.125, 0.25, 0, 0, 0]^T$	$[0.0313, 0, 0, 0.313, 0, 0, 0]^T$
2	$[.125, 0, .0625, .0625, 0, 0, 0]^T$	$[.125, 0.01, 0.08, 0.06, 0.01, 0.01, 0.02]^T$
3	$[0.0313, 0, 0, 0.0313, 0, 0, 0]^T$	$[0.25, 0.19, 0.18, 0.25, 0.02, 0.04, 0.04]^T$
4	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, .38, .15, -0.75, .08, .38, 0.4]^T$

2. *Then, we obtain $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ through Eq.(3.23) with $\mathbf{r} = \mathbf{r}^{(4)}$:*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 0, 0, 0, 1, 0, 0]^T$	$[0, .015, -0.075, 0, 0.0489, 0.0379, -0.15]^T$
1	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, -.0375, 0, 0, -.0324, 0.0075, 0]^T$
2	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, -.01, -.0187, 0]^T$
3	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, -.0062, 0, 0]^T$
4	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$

3. *After obtaining the values of $\{\mathbf{p}^{(4)}\}$ and $\{\mathbf{t}^{(4)}\}$, we derive $\Delta\mathbf{S}_e(:, c)$ using Eq.(3.22) in response to ΔG_e:*

$$\Delta\mathbf{S}_e[:, c] = \textstyle\sum_{k=0}^{4} 0.6^k \left(\mathbf{p}^{(k)}[c]\mathbf{t}^{(k)} + \mathbf{t}^{(k)}[c]\mathbf{p}^{(k)}\right)$$
$$= [0, 0, 0, 0, -.075, 0, 0]^T \quad \square$$

Leveraging the updated CoSimRank values with respect to ΔG_e, the old graph is updated by $G_1 = G_{old} \ominus \Delta G_e$, and let $\mathbf{A} = \widetilde{\mathbf{A}}$.

Then, we calculate the updated similarity scores with respect to the updating piece ΔG_f in response to query $q = c$ in the similar way.

We update the adjacency matrix with respect to ΔG_f firstly. Since edges $\langle [a, b] \rightarrow f\rangle$ are deleted from graph G_1, the in-degree of node f in the new graph $G_2 = G_1 \ominus \Delta G_f$ is equal to 0. Thus, we consider $\mathbf{w}^{(0)} = -\mathbf{A}[:, u]$ to generate $\Delta\mathbf{S}_f[:, c]$ as follows.

1. *We compute $\mathbf{w}^{(k)}$ and $\mathbf{r}^{(k)}$ using Eq.(3.24):*

k	$\mathbf{w}^{(k)}$	$\mathbf{r}^{(k)}$
0	$[-0.5, -0.5, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$
1	$[-0.25, 0, -0.25, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$
2	$[-0.125, 0, -0.125, 0, 0, 0, 0]^T$	$[-0.25, 0, -0.25, 0, 0, 0, 0]^T$
3	$[0, 0, 0, 0, 0, 0, 0]^T$	$[-0.25, -.038, -.325, 0, 0, -0.0375, -0.075]^T$
4	$[0, 0, 0, 0, 0, 0, 0]^T$	$[-0.5, -0.68, -0.075, 0, -0.03, -0.086, 0]^T$

2. *Since the in-degree of node f in G_1 is equal to δ_f, $\mathbf{t}^{(0)}$ can be calculated as $\frac{C}{2}(\mathbf{A} + \widetilde{\mathbf{A}})^T\mathbf{r}^{(K)}$.*

3. *$\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ can be iteratively generated as:*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 0, 0, 0, 0, 1, 0]^T$	$[0, -0.173, -0.15, 0, -0.1517, -.1759, 0]^T$
1	$[0, 0, 0, 0, 0.33, 0, 0]^T$	$[0, -.075, 0, 0, -.1161, 0, 0]^T$
2	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, -.025, 0, 0]^T$
3	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$
4	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$

4. *Finally, we derive $\Delta\mathbf{S}_f[:, c]$ in response to ΔG_f:*

$$\begin{aligned}\mathbf{\Delta S}_f[:, c] &= \textstyle\sum_{k=0}^{4} 0.6^k \left(\mathbf{p}^{(k)}[c]\mathbf{t}^{(k)} + \mathbf{t}^{(k)}[c]\mathbf{p}^{(k)}\right) \\ &= [0, 0, 0, 0, 0, -0.15, 0]^T \quad \square\end{aligned}$$

Theorem 4 shows that the D-deCoSim (Edge) algorithm over dynamic graph with edge deletion can efficiently and dynamically retrieve the changes of CoSimRank scores with respect to each update.

Complexity. We analyse the computational cost of D-deCoSim (Edge). We take $\tilde{n}$ as the number of nodes in the new graph $\widetilde{G} = G \ominus \Delta G$ and $\tilde{m}$ as the number of edges in the new graph. Let δ be the total number of deleted edges in ΔG and p be the number of update pieces $\{\Delta G_u\}$ in ΔG. Clearly, $p \leq \delta$. D-deCoSim has the following complexity bound:

Theorem 5. *D-deCoSim (Edge) requires $O(K(\tilde{m} + \tilde{n}p|Q|))$ time and $O(\tilde{m} + K\tilde{n})$ memory to dynamically compute $\mathbf{\Delta S}[:, Q]$ after K iterations, where $|Q|$ is the number of queries in Q.*

The computation complexity and memory complexity of the D-deCoSim (Edge) algorithm over the dynamic graphs with edge deletion is similar to the D-CoSim algorithm. Due to space restrictions, the proof of complexity is omitted here.

3.3.2.2 Similarity Search over Decreased Dynamic Graph with Node Deletion

Apart from the edge deletion in dynamic graphs, node deletion is also ubiquitous. For example, when Amazon stops selling a product, the vertex corresponding to this product needs to be removed from Amazon's selling network. Deleting a node from a graph means cutting all the in-links and out-links of the deleted node. Therefore, the similarity search algorithm over dynamic graphs with node deletion differs from the one with edge deletion. We define the similarity search algorithm over dynamic graphs with node deletion as D-deCoSim (Node). Same as before, the prior process the algorithm performs is to update the adjacency matrix.

Efficient Update of the Adjacency Matrix (Nodes' Deletion)

In dynamic graphs with node deletion, the adjacency matrix size of the new graph $\widetilde{G}$ is smaller than that of the old graph G since several nodes have been deleted from the old graph, which means the column and row of the adjacency matrix corresponding to each deleted node are deleted from the old adjacency matrix.

To efficiently update the adjacency matrix over a dynamic graph with node deletion, we first reorder the nodes of the old graph by attaching the deleted nodes to the back. Then, we separate the adjacency scores of the deleted nodes from the remaining nodes. For example, given a graph G, a node d is deleted from the graph. The old adjacency matrix is $\mathbf{A}$, which can be reorganised as $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix}$, where $\mathbf{v}$ and $\mathbf{u}^T$ are the d^{th} column and row of the old adjacency matrix. To calculate similarity score easily, we separate the CoSimRank score matrix in the same way, *e.g.,* $\mathbf{S} = \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix}$. Then, the following Lemma 3 is used to update the adjacency matrix with respect to each deleted node.

Lemma 3. *Given an old graph G and a deleted node d, the out-neighbours of node d are $\mathcal{O}(d) = \{w_i \in \mathbf{V} | (d \to w_i) \in \mathbf{E}\}$. The first step in updating the adjacency matrix is removing the d^{th} column and row from the old adjacency matrix; then, the new adjacency matrix of the new graph can be updated from the old adjacency matrix $\mathbf{A}$ by replacing the w_i^{th} column with*

$$\widetilde{\mathbf{A}}[:, w_i] = \frac{1}{1 + \delta_{w_i}/\deg_{w_i}^{-}} \mathbf{A}[:, w_i]$$

Since for each deleted node, δ_{w_i} is always equal to -1. The above equation can be rewritten as:

$$\widetilde{\mathbf{A}}[:, w_i] = \frac{\deg_{w_i}^{-}}{1 - \deg_{w_i}^{-}} \mathbf{A}[:, w_i]$$

Repeat the above equation until all $w_i^{th}(w_i^{th} \in \mathcal{O}(d))$ have been updated.

Leveraging the new adjacency matrix $\widetilde{\mathbf{A}}$ of the new graph $\widetilde{G} = G \ominus \Delta G$, we next show how to update the CoSimRank scores with respect to each node deletion dynamically. This can be achieved through several different cases. Table 3.1 summarises these cases and presents their corresponding diagram. Case 3 is the most general case, and the performance can be optimised for the special Case 1 and Case 2.

In Table 3.1, the deleted node is d, and $\mathbf{u}^T$ is the d^{th} column of the adjacent matrix, and $\mathbf{v}$ is the d^{th} row of the adjacent matrix.

Next, we detail the similarity search algorithm of these three cases.

Case 1: The deleted node only has in-neighbour nodes

From its definition, we know that CoSimRank scores are calculated using the information from in-neighbours. The deleted node not having any out-neighbour means that its information does not spread to the other nodes. In this case, the method to get the similarity scores of new graphs is illustrated in Theorem 6.

Theorem 6. *Given an old directed graph G, a query q, and a deletion update piece which can be described as G: $\Delta G(d) = ([m_1, m_2, \cdots, m_{\delta_d}] \to d) \cup (d \to [n_1, n_2, \cdots, n_{\delta_d}])$, if the out-degree of the deleted node d is zero, the CoSimRank score $\widetilde{\mathbf{S}}[:, q]$ of the new Graph in response to the query q is equal to the $\mathbf{S}_{11}[:, q]$ of the old graph.*

Proof. Firstly, the adjacency matrix $\mathbf{A}$ can be decomposed with respect to the deleted node d as $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix}$. Since the deleted node d does not have any out-neighbour, we have $\mathbf{A}_{11} = \widetilde{\mathbf{A}}$;

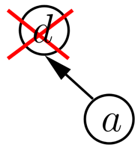
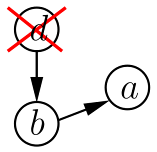
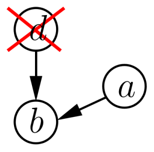
Case	Adjacency Matrix	Diagram
Case 1. The deleted node only has in-neighbour nodes ($e.g.$, node a).	$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \mathbf{0} & 0 \end{bmatrix}$	
Case 2. The deleted node has out-neighbours, and the in-degree of each out-neighbour is 1 ($e.g.$, the in-degree of node b: $deg_b^- = 1$). (Note: $\mathbf{u} = \mathbf{1}_{\{d\}}$)	$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix}$	
Case 3. The deleted node has out-neighbours, and the in-degree of some out-neighbours is larger than 1 ($e.g.$, the in-degree of node b: $deg_b^- > 1$). (Note: $b \in \mathcal{D}_d$)	$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix}$	

Table 3.1: Three Cases of Dynamic Graphs with Node Deletion

thus, the old adjacency matrix can be written as: $\mathbf{A} = \begin{bmatrix} \widetilde{\mathbf{A}} & \mathbf{v} \\ 0 & 0 \end{bmatrix}$. The CoSimRank score matrix can be decomposed by the same method: $\mathbf{S} = \begin{bmatrix} \mathbf{S}_{11} & (\mathbf{S}_{21})^T \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix}$.

The basic similarity search algorithm is as follows.

$$\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I}$$

Substitute the new format of $\mathbf{A}$ and $\mathbf{S}$ into the algorithm:

$$\mathbf{S} = C \begin{bmatrix} \widetilde{\mathbf{A}}^T & 0 \\ \mathbf{v}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{S}_{11} & (\mathbf{S}_{21})^T \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} & \mathbf{v} \\ 0 & 0 \end{bmatrix} + \mathbf{I}$$

According to the basic similarity search algorithm, $\mathbf{S}_{11}$ and $\widetilde{\mathbf{A}}$ can be calculated as,

$$\mathbf{S}_{11} = C\widetilde{\mathbf{A}}^T\mathbf{S}_{11}\widetilde{\mathbf{A}} + \mathbf{I}$$

$$\widetilde{\mathbf{S}} = C\widetilde{\mathbf{A}}^T\widetilde{\mathbf{S}}\widetilde{\mathbf{A}} + \mathbf{I}$$

Then, $\Delta\mathbf{S}$ can be generated by,

$$\Delta\mathbf{S} = \widetilde{\mathbf{S}} - \mathbf{S}_{11} = C\widetilde{\mathbf{A}}^T(\widetilde{\mathbf{S}} - \mathbf{S}_{11})\widetilde{\mathbf{A}} \equiv \mathbf{0}$$

Thus, for query q, we can conclude that $\widetilde{\mathbf{S}}[:, q] = \mathbf{S}_{11}[:, q]$. □

The proof illustrates the correctness of Theorem 6. Next, we discuss the complexity of D-deCoSim (Node) Case 1 as follows.

Complexity. We now analyse the computational cost of D-deCoSim (Node) for Case 1 (the deleted node does not have out-neighbours). Let δ denote the number of deleted nodes in the new graph and $\tilde{n}$ and $\tilde{m}$ represent the number of nodes and edges in the new graph, respectively. The computation complexity bound of the algorithm is shown in Theorem 7.

Theorem 7. *D-deCoSim (Node) over dynamic graph with node deletion (Case 1: the deleted node without out-neighbours) requires $O(\delta|Q|)$ time and $O(\tilde{n} + \tilde{m})$ memory to get $\mathbf{S}[:, Q]$, where $|Q|$ is the number of queries.*

Then, we take an example to show how Theorem 6 updates the CoSimRank scores of a new graph in response to a query.

Example 7. *In Figure 3.6, given an old graph G with 4 nodes and the deleted node d, d does not have any out-neighbours. There are 3 nodes in the new graph $\widetilde{G} = G \ominus \Delta G$. We take node a as query and generate the similarity score of $\widetilde{\mathbf{S}}[:, a]$ using Theorem 6 as follows.*

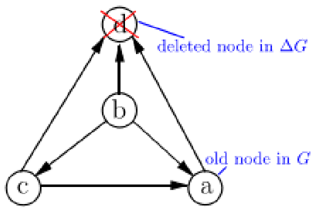


Figure 3.6: Case 1: Nodes Deletion ΔG of Graph G

Before computing the CoSimRank scores, the adjacency matrix should be updated with respect to the deleted node. Therefore, we remove the 4^{th} *column and* 4^{th} *row, which correspond to node* d*, from the old adjacency matrix:*

$$\mathbf{A} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 1 & 1 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \rightarrow \widetilde{\mathbf{A}} = \begin{bmatrix} 0 & 0 & 0 \\ \frac{1}{2} & 0 & 1 \\ \frac{1}{2} & 0 & 0 \end{bmatrix}$$

Following Theorem 6 and the known similarity scores of the old graph, the similarity scores of the new graph $\widetilde{G}$ *in response to query* a *is as follows:*

$$\mathbf{S}[:,a] = [1.39, 0, 0.3, 0.3]^T \quad \rightarrow \quad \mathbf{S}_{11}[:,a] = \widetilde{\mathbf{S}}[:,a] = [1.39, 0, 0.3]^T$$

Case 2: The deleted node has out-neighbours, and the in-degree of each out-neighbour equal to 1.

In this case, the deleted node d has out-neighbours, and the in-degree of each out-neighbour equals 1 in the old graph G. Thus, the out-neighbours $O(d)$ of the deleted node d only have one in-neighbour in the old graph G, which is the deleted node d. Therefore, the similarity information of the out-neighbours $O(d)$ are only capture the information from the deleted node d. The similarity scores of the new graph with respect to the deleted node d can be generated using Theorem 8.

Theorem 8. *Given a directed graph* G*, a query* q*, and deleted node* d *from the graph, the in-degree of the node* d*'s out-neighbours are equal to 1s (Case 2); the changes in the CoSimRank score* $\Delta\mathbf{S}[:,q]$ *with respect to query* q *are dynamically computed as,*

$$\Delta\mathbf{S}[:,q] = \sum_{k=0}^{\infty} C^k(\mathbf{p}^{(k)}[q](\mathbf{t}^{(k)}) + \mathbf{t}^{(k)}[q](\mathbf{p}^{(k)})) \tag{3.25}$$

where $\mathbf{p}^{(k)}$ *and* $\mathbf{t}^{(k)}$ *are iteratively obtained by,*

$$\begin{cases} \mathbf{p}^{(0)} = \mathbf{u} \\ \\ \mathbf{p}^{(k)} = \widetilde{\mathbf{A}}^T \mathbf{p}^{(k-1)} \end{cases} \qquad \begin{cases} \mathbf{t}^{(0)} = (-C)(\widetilde{\mathbf{A}}\mathbf{S}_{12} + \frac{C}{2}\mathbf{u}\mathbf{S}_{22}) \\ \\ \mathbf{t}^{(k)} = \widetilde{\mathbf{A}}^T \mathbf{t}^{(k-1)} \end{cases} \tag{3.26}$$

where $\mathbf{u} = \mathbf{A}[d,i]^T$ *(*$i \in \mathbf{V} | i \neq d$*).* $\mathbf{S}_{12}$ *is the* d^{th} *column of the old CoSimRank score matrix except for the value of* $\mathbf{S}(d,d)$*, and* $\mathbf{S}_{22}$ *is the value of* $\mathbf{S}(d,d)$*.*

Proof. Given an old directed graph G and a deleted node d from the graph, each out-neighbour of the node d only has one in-neighbour.

As mentioned previously, the adjacency matrix is column-normalised; thus, the values in each column are determined by the number of in-neighbours of the corresponding node. The adjacency matrix of old graph G is the following:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{A}} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix}$$

Since the in-degree of the deleted node d's out-neighbours $\mathcal{O}(d)$ are equal to 1s in old graph G, the deleted node d will not affect the remaining values of the old adjacency matrix, which means $\mathbf{A}_{11} = \widetilde{\mathbf{A}}$.

Then, substitute $\mathbf{A}$ to the basic similarity search algorithm:

$$\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I}$$

$$\mathbf{S} = C \begin{bmatrix} \widetilde{\mathbf{A}}^T & 0 \\ \mathbf{v}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{S}_{11} & (\mathbf{S}_{21})^T \\ \mathbf{S}_{21} & \mathbf{S}_{22} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} & \mathbf{v} \\ 0 & 0 \end{bmatrix} + \mathbf{I}$$

where $\mathbf{u} = \mathbf{A}[d,i]^T$ $(i \in \mathbf{V} | i \neq d)$.

Then, $\mathbf{S}_{11}$ can be generated by

$$\mathbf{S}_{11} = C(\widetilde{\mathbf{A}}^T\mathbf{S}_{11} + \mathbf{u}{\mathbf{S}_{12}}^T)\widetilde{\mathbf{A}} + (\widetilde{\mathbf{A}}^T\mathbf{S}_{12} + \mathbf{u}\mathbf{S}_{22})\mathbf{u}^T + \mathbf{I}$$

and according to the basic similarity score algorithm, $\widetilde{\mathbf{S}}$ is

$$\widetilde{\mathbf{S}} = C\mathbf{A}^T(\mathbf{S}_{11} + \Delta\mathbf{S})\mathbf{A} + \mathbf{I}$$

where $\Delta\mathbf{S}$ can be calculated as

$$\Delta\mathbf{S} = C\widetilde{\mathbf{A}}^T\Delta\mathbf{S}\widetilde{\mathbf{A}} - C\mathbf{u}{\mathbf{S}_{12}}^T\widetilde{\mathbf{A}} - C\widetilde{\mathbf{A}}^T\mathbf{S}_{12}\mathbf{u}^T - C\mathbf{u}\mathbf{S}_{22}\mathbf{u}^T \tag{3.27}$$

Here, we set $-C \cdot (\mathbf{u}\mathbf{S}_{12}{}^T\widetilde{\mathbf{A}} + \widetilde{\mathbf{A}}^T\mathbf{S}_{12}\mathbf{u}^T + \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T)$ as $\mathbf{E}$, and $\mathbf{f}_u = \widetilde{\mathbf{A}}^T\mathbf{S}_{12}$,

$$\begin{aligned}
\mathbf{E} &= -C \cdot (\mathbf{u}\mathbf{S}_{12}{}^T\widetilde{\mathbf{A}} + \widetilde{\mathbf{A}}^T\mathbf{S}_{12}\mathbf{u}^T + \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T) \quad \textit{let } \mathbf{f}_u = \widetilde{\mathbf{A}}^T\mathbf{S}_{12} \\
&= -C \cdot (\mathbf{u}\mathbf{f}_u{}^T + \mathbf{f}_u\mathbf{u}^T + \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T) \\
&= -C \cdot (\mathbf{u}(\mathbf{f}_u{}^T + \frac{1}{2}\mathbf{S}_{22}\mathbf{u}^T) + (\mathbf{f}_u + \frac{1}{2}\mathbf{u}\mathbf{S}_{22})\mathbf{u}^T) \quad \textit{let } \mathbf{x} = \mathbf{f}_u + \frac{1}{2}\mathbf{u}\mathbf{S}_{22} \\
&= -C \cdot (\mathbf{u}\mathbf{x}^T + \mathbf{x}\mathbf{u}^T)
\end{aligned}$$

Substitute $\mathbf{E} = -C \cdot (\mathbf{u}\mathbf{x}^T + \mathbf{x}\mathbf{u}^T)$ into Eq.(3.27),

$$\begin{aligned}
\Delta\mathbf{S} &= \sum\nolimits_{k=0}^{\infty} C^k (\widetilde{\mathbf{A}}^T)^k \mathbf{E} (\widetilde{\mathbf{A}})^k \\
&= \sum\nolimits_{k=0}^{\infty} C^k (\widetilde{\mathbf{A}}^T)^k (-C)(\mathbf{u}\mathbf{x}^T + \mathbf{x}\mathbf{u}^T)(\widetilde{\mathbf{A}})^k \\
&= -C \cdot \sum\nolimits_{k=0}^{\infty} C^k ((\widetilde{\mathbf{A}}^T)^k \mathbf{u}\mathbf{x}^T (\widetilde{\mathbf{A}})^k + ((\widetilde{\mathbf{A}}^T)^k \mathbf{x}\mathbf{u}^T (\widetilde{\mathbf{A}})^k)
\end{aligned}$$

Finally, let $\mathbf{t}^{(k)} = -C \cdot (\widetilde{\mathbf{A}}^T)^k \mathbf{x}$, $\mathbf{p}^{(k)} = (\widetilde{\mathbf{A}}^T)^k \mathbf{u}$, so $\Delta\mathbf{S}$ can be generated as

$$\Delta\mathbf{S} = \sum\nolimits_{k=0}^{\infty} C^k (\mathbf{p}^{(k)}(\mathbf{t}^{(k)}) + \mathbf{t}^{(k)}(\mathbf{p}^{(k)}))$$

$\square$

Complexity. The above proof clearly establishes the correctness of the similarity search algorithm over the dynamic graph with node deletion (Case 2: the in-degree of the deleted node's out-neighbours is equal to 1). Based on Theorem 8, we now analyse the computational cost of the algorithm. δ is the number of the deleted nodes from the old graph G, $\mathbf{d} = [d_1, d_2, \cdots, d_\delta]$ is the list of deleted nodes, $\widetilde{n}$ and $\widetilde{m}$ are respectively the number of nodes and edges in the new graph, and p is the number of updated bunches of ΔG, where $p = \sum_{d_i \in \mathbf{d}}^{\delta} deg_{d_i}^{+}$.

The computation complexity of the algorithm can be expressed as follows.

Theorem 9. *Given a directed graph, a bunch of deleted nodes that satisfied the condition in case 2, a query set Q, and the number of deleted nodes from the graph G is $\widetilde{n}$, $O(K(\widetilde{m} + \widetilde{n}p|Q|))$ time and $O(\widetilde{m} + K\widetilde{n})$ memory are required to get $\Delta\mathbf{S}[:, Q]$.*

Here, time $O(K\widetilde{m})$ comes from matrix-vector products, which include $\widetilde{\mathbf{A}}^T\mathbf{p}^{(k-1)}$ and $\widetilde{\mathbf{A}}^T\mathbf{t}^{(k-1)}$ in Eq. 3.26. The memory complexity $O(\widetilde{m} + K\widetilde{n})$ comes from the storage of adjacency matrix, rather than $\mathbf{p}^i$ and $\mathbf{t}^i$, since all values of $\mathbf{p}^i$ and $\mathbf{t}^i$ are deleted when one update computation is finished.

Next, we take an example to show how Theorem 8 updates similarity scores of a decremental dynamic graph with node deletion (Case 2).

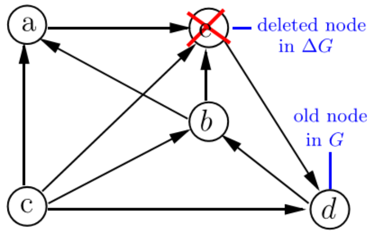


Figure 3.7: Case 2 of Decremental Dynamic Graphs with Node Deletion

Example 8. *In Figure 3.7, the old graph G has five nodes, and the deleted node is e (a node with red cross). Given a query c, number of iterations $K = 4$, and damping factor $C = 0.6$, we follow Theorem 8 to dynamically retrieve $\Delta\mathbf{S}[:, q]$ through the following steps.*

There are five nodes in graph G. To get the updated adjaceny matrix, the e^{th} column and e^{th} row of the old adjacency matrix are removed. The new adjacency matrix can be described as follows: $\mathbf{A} = \begin{bmatrix} 0 & 0 & \frac{1}{2} & 0 & \frac{1}{2} \\ 1 & 0 & \frac{1}{2} & 0 & 0 \\ 0 & 0 & 0 & 0 & \frac{1}{2} \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \rightarrow \widetilde{\mathbf{A}} = \begin{bmatrix} 0 & 0 & \frac{1}{2} & 0 \\ 1 & 0 & \frac{1}{2} & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$

Following the similarity search algorithm, the similarity scores of the old graph is

$$\mathbf{S} = \begin{bmatrix} 2.3524 & 0.0532 & 0.6922 & 0.0018 & 0.0241 \\ 0.0532 & 2.2540 & 0.0271 & 0.0887 & 0.0030 \\ 0.6922 & 0.0271 & 1.7069 & 0.0081 & 0.4687 \\ 0.0018 & 0.0887 & 00081 & 2.0899 & 0.1479 \\ 0.0241 & 0.0030 & 04687 & 0.1479 & 1.8165 \end{bmatrix}.$$

According to Eq.(3.26)*, we have* $\mathbf{u} = [\,0\ 0\ 0\ 1\,]^T$*; thus,* $\{\mathbf{p}^{(k)}\}$ *and* $\{\mathbf{t}^{(k)}\}$ *can be iteratively generated as*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 0, 0, 1]^T$	$[-0.0018, -0.0887, -0.0081, -0.545]^T$
1	$[0, 0, 0, 1]^T$	$[-0.0887, -0.0545, -0.452, 0]^T$
2	$[1, 0, 0.5, 0]^T$	$[-0.545, 0, -0.3168, 0]^T$
3	$[0, 0, 0.5, 0]^T$	$[0, 0, -0.2725, 0]^T$
4	$[0, 0, 0, 0]^T$	$[0, 0, 0, 0]^T$

Substituting the results of $\{\mathbf{p}^{(k)}\}$ *and* $\{\mathbf{t}^{(k)}\}$ *into Eq.*(3.25)*,*

$$\Delta\mathbf{S}[:, c] = \sum\nolimits_{k=0}^{4} 0.6^4(\mathbf{p}^{(4)}[c](\mathbf{t}^{(4)}) + \mathbf{t}^{(4)}[c](\mathbf{p}^{(4)})) = [-0.2122, -0.0271, -0.1729, -0.0081]^T$$

Finally, according to $\widetilde{\mathbf{S}} = \mathbf{S} + \Delta\mathbf{S}$*, the CoSimRank score in response to query c of the new graph is*

$$\widetilde{\mathbf{S}}[:, c] = \mathbf{S}[:, c] + \Delta\mathbf{S}[:, c] = [0.48, 0, 1.534, 0]^T$$

Case 3: The deleted node has out-neighbours, and the in-degree of some out-neighbours is larger than 1.

The case covers the remaining cases of nodes being deleted from a graph. Since the algorithm generates the similarity scores by getting information from the in-neighbours of a node pair, when the in-degree of some nodes $\mathcal{O}(d)$ are larger than 1, the new adjacency matrix is not equal to $\mathbf{A}_{11}$. Therefore, we need to update the adjacency matrix using Lemma 3.

Leveraging the new adjacency matrix, we next show the algorithm to calculate CoSimRank scores in response to query q of the new graph.

Theorem 10. *Given an old directed graph G, the node d is deleted from the graph. The in-degree of some of node d's out-neighbours are larger than 1 in the old graph(Case 3). Here, we denote the nodes in $\mathcal{O}(d)$ whose in-degree is larger than 1 as $\mathcal{D}_{out}$; then, the changes $\Delta\mathbf{S}[:,q]$ to the CoSimRank scores are dynamically computed as*

$$\Delta\mathbf{S}[:,q] = \sum\nolimits_{k=0}^{\infty} C^k(\mathbf{p}^{(k)}[q](\mathbf{t}^{(k)}) + \mathbf{t}^{(k)}[q](\mathbf{p}^{(k)}) + \mathbf{w}^{(k)}[q](\mathbf{r}^{(k)}) + \mathbf{r}^{(k)}[q](\mathbf{w}^{(k)})) \tag{3.28}$$

where $\mathbf{p}^{(k)}[q]$ and $\mathbf{t}^{(k)}[q]$ are the q^{th} entry of the vectors $\mathbf{p}^{(k)}$ and $\mathbf{t}^{(k)}$ respectively, and they are calculated using the following:

$$\begin{cases}\mathbf{p}^{(0)} = \mathbf{e}_{\mathcal{D}_{out}} \\ \mathbf{p}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{p}^{(k-1)}\end{cases} \quad \begin{cases}\mathbf{t}^{(0)} = \dfrac{C}{2}\cdot\mathbf{f} \\ \mathbf{t}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{t}^{(k-1)}\end{cases} \tag{3.29}$$

Similarly, $\mathbf{w}^{(k)}[q]$ and $\mathbf{r}^{(k)}[q]$ can be iteratively obtained by

$$\begin{cases}\mathbf{r}^{(0)} = -\mathbf{u} \\ \mathbf{r}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{r}^{(k-1)}\end{cases} \quad \begin{cases}\mathbf{w}^{(0)} = C\cdot({\mathbf{A}_{11}}^T\mathbf{S}_{12} + \dfrac{1}{2}\mathbf{u}\mathbf{S}_{22}) \\ \mathbf{w}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{w}^{(k-1)}\end{cases} \tag{3.30}$$

where $\mathbf{e}_{\mathcal{D}_{out}}$ is a unit vector with 1 in the $\mathcal{D}_{out}^{th}$ entry.

For the calculation of $\mathbf{t}^{(0)}$ in Equation 3.29,

$$\mathbf{f} = (\mathbf{A}_{11} + \widetilde{\mathbf{A}})^T\mathbf{S}_{11}(\mathbf{A}_{11}(:,D_{out}) + \frac{1}{\deg^-_{D_{out}}\cdot(\deg^-_{D_{out}}-1)}$$

$\mathbf{A}_{11}$ is extracted from the old adjacency matrix by removing the d^{th} column and row of the old adjacency matrix $\mathbf{A}$. Similarly, $\mathbf{S}_{11}$ is extracted from the CoSimRank scores of the old graph by removing the d^{th} column and row of the old CoSimRank score matrix $\mathbf{S}$.

Proof. Given an old directed graph G and a node d deleted from the graph, the in-degree of some of node d's out-neighbours is not equal to 0 in the new graph $\widetilde{G}$. Thus, the adjacency matrix of old graph G is

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \\ \mathbf{u}^T & 0 \end{bmatrix} = \begin{bmatrix} \widetilde{\mathbf{A}} + \Delta\mathbf{A} & \mathbf{v} \\ \\ \mathbf{u}^T & 0 \end{bmatrix}$$

then, substitute A into the basic similarity search algorithm:

$$\begin{aligned} \mathbf{S} &= C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I} \\ &= C \cdot \begin{bmatrix} (\widetilde{\mathbf{A}} - \Delta\mathbf{A})^T & \mathbf{u} \\ \mathbf{v}^T & 0 \end{bmatrix} \begin{bmatrix} \mathbf{S}_{11} & \mathbf{S}_{12} \\ \mathbf{S}_{12}{}^T & \mathbf{S}_{22} \end{bmatrix} \begin{bmatrix} \widetilde{\mathbf{A}} - \Delta\mathbf{A} & \mathbf{v} \\ \mathbf{u}^T & 0 \end{bmatrix} + \mathbf{I} \end{aligned}$$

Then, $\mathbf{S}_{11}$ can be generated as

$$\mathbf{S}_{11} = C(((\widetilde{\mathbf{A}} - \Delta\mathbf{A})^T\mathbf{S}_{11} + \mathbf{u}\mathbf{S}_{12}{}^T)(\widetilde{\mathbf{A}} - \Delta\mathbf{A}) + ((\widetilde{\mathbf{A}} - \Delta\mathbf{A})^T\mathbf{S}_{12} + \mathbf{u}\mathbf{S}_{22})\mathbf{u}^T) + \mathbf{I}$$

According to the basic similarity score algorithm, $\widetilde{\mathbf{S}}$ is

$$\widetilde{\mathbf{S}} = C\widetilde{\mathbf{A}}^T(\mathbf{S}_{11} + \Delta\mathbf{S})\widetilde{\mathbf{A}} + \mathbf{I}$$

where $\Delta\mathbf{S}$ can be calculated as follows:

$$\begin{aligned} \Delta\mathbf{S} = C\widetilde{\mathbf{A}}^T\Delta\mathbf{S}\widetilde{\mathbf{A}} - C \cdot (&\mathbf{u}\mathbf{S}_{12}{}^T(\widetilde{\mathbf{A}} - \Delta\mathbf{A}) + (\widetilde{\mathbf{A}} - \Delta\mathbf{A})^T\mathbf{S}_{12}\mathbf{u}^T \\ &+ \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T - (\widetilde{\mathbf{A}} - \Delta\mathbf{A})^T\mathbf{S}_{11}\Delta\mathbf{A} - \Delta\mathbf{A}^T\mathbf{S}_{11}\widetilde{\mathbf{A}}) \end{aligned}$$

Then, we set

$$\begin{aligned} \mathbf{E} &= -C \cdot (\mathbf{u}\mathbf{S}_{12}{}^T(\widetilde{\mathbf{A}} - \Delta\mathbf{A}) + (\widetilde{\mathbf{A}} - \Delta\mathbf{A})^T\mathbf{S}_{12}\mathbf{u}^T \\ &\quad + \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T - (\widetilde{\mathbf{A}} - \Delta\mathbf{A})^T\mathbf{S}_{11}\Delta\mathbf{A} - \Delta\mathbf{A}^T\mathbf{S}_{11}\widetilde{\mathbf{A}}) \\ &\mathit{here}\ \widetilde{\mathbf{A}} - \Delta\mathbf{A} = \mathbf{A}_{11} \\ &\quad = -C \cdot (\mathbf{u}\mathbf{S}_{12}{}^T\mathbf{A}_{11} + \mathbf{A}_{11}{}^T\mathbf{S}_{12}\mathbf{u}^T + \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T - \mathbf{A}_{11}{}^T\mathbf{S}_{11}\Delta\mathbf{A} - \Delta\mathbf{A}^T\mathbf{S}_{11}\widetilde{\mathbf{A}}) \\ &\mathit{here}\ \widetilde{\mathbf{A}} = \mathbf{A}_{11} + \Delta\mathbf{A}[:, \mathcal{D}_{out}]\mathbf{e}_{\mathcal{D}_{out}}^T \\ &\quad = -C \cdot (\mathbf{u}\mathbf{S}_{12}{}^T\mathbf{A}_{11} + \mathbf{A}_{11}{}^T\mathbf{S}_{12}\mathbf{u}^T + \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T - \mathbf{A}_{11}{}^T\mathbf{S}_{11}\Delta\mathbf{A}[:, \mathcal{D}_{out}]\mathbf{e}_{\mathcal{D}_{out}}^T \\ &\qquad - (\Delta\mathbf{A}[:, \mathcal{D}_{out}]\mathbf{e}_{\mathcal{D}_{out}}^T)^T\mathbf{S}_{11}(\mathbf{A}_{11} + \Delta\mathbf{A}[:, \mathcal{D}_{out}]\mathbf{e}_{\mathcal{D}_{out}}^T)) \end{aligned}$$

After describing the new adjacency matrix, we set f as follows:

$$\begin{aligned}
\mathbf{f} &= \mathbf{S}_{11}\frac{1}{\deg^-(\mathcal{D}_{out})\cdot(\deg^-(\mathcal{D}_{out})+1)}\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}} \\
&= \frac{1}{\deg^-(\mathcal{D}_{out})\cdot(\deg^-(\mathcal{D}_{out})+1)}\big(\textstyle\sum_{k=0}^{\infty} C^k(\mathbf{h}^{(k)}(\mathbf{m}^{(k)})^T \\
&\qquad + (\mathbf{h}^{(k)})^T\mathbf{m}^{(k)} + (\mathbf{A}_{11}{}^T)^k\mathbf{A}_{11}{}^k)\big)\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}} \\
&= \frac{1}{\deg^-(\mathcal{D}_{out})\cdot(\deg^-(\mathcal{D}_{out})+1)}\big(\underbrace{(\textstyle\sum_{k=0}^{\infty} C^k(\mathbf{h}^{(k)}(\mathbf{m}^{(k)})^T + (\mathbf{h}^{(k)})^T\mathbf{m}^{(k)}))\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}}}_{\mathbf{y}^{(K)}} \\
&\qquad + \underbrace{\textstyle\sum_{k=0}^{\infty} C^k(\mathbf{A}_{11}{}^T)^k\mathbf{A}_{11}{}^k\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}}}_{\mathbf{j}^{(K)}}\big)
\end{aligned}$$

$$\begin{aligned}
&let\ \mathbf{f}_x = \mathbf{S}_{11}\Delta\mathbf{A}[:,\mathcal{D}_{out}],\quad \mathbf{f}_y = \mathbf{A}_{11}{}^T\mathbf{S}_{12} \\
&\quad = -C\cdot(\mathbf{u}\mathbf{f}_y^T + \mathbf{f}_y\mathbf{u}^T + \mathbf{u}\mathbf{S}_{22}\mathbf{u}^T - \mathbf{A}_{11}{}^T\mathbf{f}_x\mathbf{e}_{\mathcal{D}_{out}}^T - \mathbf{e}_{\mathcal{D}_{out}}\ \mathbf{f}_x^T\mathbf{A}_{11} - \mathbf{e}_{\mathcal{D}_{out}}\Delta\mathbf{A}[:,\mathcal{D}_{out}]^T\mathbf{f}_x\mathbf{e}_{\mathcal{D}_{out}}^T) \\
&\quad = (-C)(\mathbf{u}(\mathbf{f}_y^T + \frac{1}{2}\mathbf{S}_{22}\mathbf{u}^T) + (\mathbf{f}_y + \frac{1}{2}\mathbf{u}\mathbf{S}_{22})\mathbf{u}^T - (\mathbf{A}_{11}{}^T\mathbf{f}_x + \frac{1}{2}\mathbf{e}_{\mathcal{D}_{out}}\Delta\mathbf{A}[:,\mathcal{D}_{out}]^T\mathbf{f}_x)\mathbf{e}_{\mathcal{D}_{out}}^T \\
&\qquad - \mathbf{e}_{\mathcal{D}_{out}}(\mathbf{f}_x^T\mathbf{A}_{11} + \frac{1}{2}\Delta\mathbf{A}[:,\mathcal{D}_{out}]^T\mathbf{f}_x\mathbf{e}_{\mathcal{D}_{out}}^T)) \\
&let\ \mathbf{x} = C\mathbf{A}_{11}{}^T\mathbf{f}_x + \frac{C}{2}\mathbf{e}_{\mathcal{D}_{out}}\mathbf{f}_x^T\Delta\mathbf{A}[:,\mathcal{D}_{out}]^T\mathbf{f}_x,\ \mathbf{y} = C\mathbf{f}_y + \frac{C}{2}\mathbf{u}\mathbf{S}_{22} \\
&\quad = \mathbf{x}\mathbf{e}_{\mathcal{D}_{out}}^T + \mathbf{e}_{\mathcal{D}_{out}}\mathbf{x}^T - \mathbf{y}\mathbf{u}^T - \mathbf{u}\mathbf{y}^T
\end{aligned}$$

Substitute x into the basic iterative similarity search algorithm of $\Delta\mathbf{S}$ with respect to δG,

$$\begin{aligned}
\Delta\mathbf{S} &= \textstyle\sum_{k=0}^{\infty} C^k(\widetilde{\mathbf{A}}^T)^k\mathbf{E}(\widetilde{\mathbf{A}})^k \\
&= \textstyle\sum_{k=0}^{\infty} C^k(\widetilde{\mathbf{A}}^T)^k(\mathbf{x}\mathbf{e}_{\mathcal{D}_{out}}^T + \mathbf{e}_{\mathcal{D}_{out}}\mathbf{x}^T - \mathbf{y}\mathbf{u}^T - \mathbf{u}\mathbf{y}^T)\widetilde{\mathbf{A}}^k
\end{aligned}$$

Let $\mathbf{t}^{(k)} = (\widetilde{\mathbf{A}}^T)^k\mathbf{x},\quad \mathbf{p}^{(k)} = (\widetilde{\mathbf{A}}^T)^k\mathbf{e}_{\mathcal{D}_{out}}$, where

$$\begin{aligned}
\mathbf{x} &= C\mathbf{A}_{11}{}^T\mathbf{f}_x + \frac{C}{2}\mathbf{e}_{\mathcal{D}_{out}}\mathbf{f}_x^T\Delta\mathbf{A}[:,\mathcal{D}_{out}]^T\mathbf{f}_x \\
&= \frac{C}{2}(\mathbf{A}_{11} + \widetilde{\mathbf{A}})^T\mathbf{S}_{11}\Delta\mathbf{A}[:,\mathcal{D}_{out}]
\end{aligned}$$

Next we set $\mathbf{m}^{(k)} = -(\widetilde{\mathbf{A}}^T)^k\mathbf{y},\quad \mathbf{n}^{(k)} = -(\widetilde{\mathbf{A}}^T)^k\mathbf{u}$. Here, $\Delta\mathbf{A}(:,\mathcal{D}_{out})$ can be calculated by $\Delta\mathbf{A}(:,\mathcal{D}_{out}) = \frac{1}{\deg^-(\mathcal{D}_{out})\cdot(\deg^-(\mathcal{D}_{out})+1)}\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}}$.

Thus, $\Delta\mathbf{S}$ can be generated as follows:

$$\Delta\mathbf{S} = \sum_{k=0}^{\infty} C^k(\mathbf{p}^{(k)}(\mathbf{t}^{(k)})^T + \mathbf{t}^{(k)}(\mathbf{p}^{(k)})^T + \mathbf{w}^{(k)}(\mathbf{r}^{(k)})^T + \mathbf{r}^{(k)}(\mathbf{w}^{(k)})^T)$$

$$where \begin{cases} \mathbf{p}^{(0)} = \mathbf{e}_{\mathcal{D}_{out}} \\ \mathbf{p}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{p}^{(k-1)} \end{cases} \quad \begin{cases} \mathbf{t}^{(0)} = \frac{C}{2}(\mathbf{A}_{11} + \widetilde{\mathbf{A}})^T \cdot \mathbf{f} \\ \mathbf{t}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{t}^{(k-1)} \end{cases}$$

$$\begin{cases} \mathbf{r}^{(0)} = -\mathbf{u} \\ \mathbf{r}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{r}^{(k-1)} \end{cases} \quad \begin{cases} \mathbf{w}^{(0)} = C \cdot (\mathbf{A}_{11}{}^T\mathbf{S}_{12} + \frac{1}{2}\mathbf{u}\mathbf{S}_{22}) \\ \mathbf{w}^{(k)} = \widetilde{\mathbf{A}}^T\mathbf{w}^{(k-1)} \end{cases}$$

$$here\ \mathbf{f} = \mathbf{S}_{11}\frac{1}{\deg^-(\mathcal{D}_{out}) \cdot (\deg^-(\mathcal{D}_{out}) + 1)}\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}}$$

□

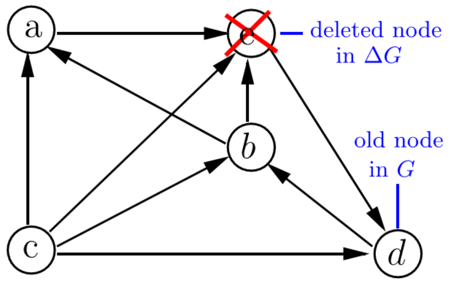


Figure 3.8: Case 3: Node Deletion ΔG of Graph G

Next, we take an example to show how the D-deCoSim (Node) algorithm (Case 3) retrieves the similarity scores of "refreshed area".

Example 9. *Given a directed graph G with five nodes and a deleted node e (a node with red cross) (Figure 3.8), set query $q = b$, the number of iterations $K = 3$, and decay factory $C = 0.6$; then, we iteratively retrieve $\Delta\mathbf{S}[:, q]$ as follows.*

Figure 3.8 shows that the deleted node e has one out-neighbour (node d). Node d has 2 in-neighbours in the old graph G (node e and c), so the node d belongs to $\mathcal{D}_{out}$. This example satisfies the conditions of Case 3; thus, Theorem 10 is used to iteratively retrieve $\Delta\mathbf{S}[:, q]$.

Firstly, Lemma 3 is used to update the adjacency matrix in response to the deletion of node d.

$$\widetilde{\mathbf{A}}(:, \mathcal{D}_{out}) = \mathbf{A}_{11}(:, \mathcal{D}_{out}) + \frac{1}{\deg^-(\mathcal{D}_{out})(\deg^-(\mathcal{D}_{out}) - 1)}\mathbf{1}_{\mathcal{I}(\mathcal{D}_{out})}$$
$$= \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \\ 0 \end{bmatrix} + \frac{1}{1 * 2}\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

The remaining entries of the old adjacency matrix are the same as before. Therefore, the new adjacency matrix is

$$\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{v} \\ \mathbf{u} & 0 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & 0 & 0 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} & 0 \\ 0 & \frac{1}{2} & 0 & 0 & 0 \\ 0 & 0 & 1 & \frac{1}{2} & 0 \end{bmatrix} \quad \rightarrow \quad \widetilde{\mathbf{A}} = \begin{bmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 1 & 0 \\ 0 & \frac{1}{2} & 0 & 0 \end{bmatrix}$$

The CoSimRank scores matrix with respect to the query b of the old graph G is

$$\mathbf{S}[:, b] = \begin{bmatrix} 0.4484, & 1.6929, & 0.0722, & 0.4119, & 0.1377 \end{bmatrix}^T$$

Substituting $u = [0, \quad 0, \quad 0, \quad \frac{1}{2}]^T$, $\mathbf{S}_{11}$ and $\mathbf{S}_{22}$ into Eq. 3.30 to compute $\mathbf{w}^{(K)}$ and $\mathbf{r}^{(K)}$, we can get

k	$\mathbf{r}^{(k)}$	$\mathbf{w}^{(k)}$
0	$[0, 0, 0, 1]^T$	$[0.15, 0.15, 0, 0.225]^T$
1	$[0, 0.5, 0, 0]^T$	$[0.075, 0.1125, 0, 0]^T$
2	$[0.25, 0, 0, 0]^T$	$[0.0562, 0, 0, 0]^T$
3	$[0, 0, 0, 0]^T$	$[0, 0, 0, 0]^T$

Next, we obtain $\mathbf{p}^{(K)}$ and $\mathbf{t}^{(K)}$ using Eq. 3.29:

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 0, 0, -0.5]^T$	$[0.0402, 0.0416, 0, 0.1905]^T$
1	$[0, -0.25, 0, 0]^T$	$[0.0208, 0.0953, 0, 0]^T$
2	$[-0.125, 0, 0, 0]^T$	$[0.0476, 0, 0, 0]^T$
3	$[0, 0, 0, 0]^T$	$[0, 0, 0, 0]^T$

Then, the updated parts of S with respect to the "refreshed area" is

$$\Delta\mathbf{S}[:, b] = \sum_{k=0}^{3} C^3 (\mathbf{p}^{(3)}[b](\mathbf{t}^{(3)})^T + \mathbf{t}^{(3)}[b](\mathbf{p}^{(3)})^T + \mathbf{w}^{(3)}[b](\mathbf{r}^{(3)})^T + \mathbf{r}^{(3)}[b](\mathbf{w}^{(3)})^T)$$
$$= \begin{bmatrix} 0.0194 & 0.0389 & 0 & 0.1292 \end{bmatrix}^T$$

Finally, the CoSimRank scores of the new graph with respect to query q is

$$\widetilde{\mathbf{S}}[:, q] = \Delta\mathbf{S}[:, q] + \mathbf{S}_{11}[:, q] = \begin{bmatrix} 0.195 & 1.39 & 0 & 0.3 \end{bmatrix}^T$$

Complexity. In case 3(the deleted node has out-neighbours, and the in-degree of some out-neighbours is larger than 1), the complexity of the similarity search computation is closely related to the number of $\mathcal{D}_{out}$. Here, we define $|\mathcal{D}_{out}|$ as the number of $\mathcal{D}_{out}$. $\widetilde{n}$ and $\widetilde{m}$ denote the number of nodes and edges of the new graph, respectively. According to Theorem 10, $\widetilde{n}$ is the number of deleted nodes of the old graph G. Hence, the computation complexity of the algorithm can be expressed as

Theorem 11. *Given a directed graph, a bunch of deleted nodes, the number of deleted nodes as $\widetilde{n}$, and a query set Q, are required $O(K(\widetilde{m} + |Q||\mathcal{D}_{out}|\widetilde{n}))$ time and $O(\widetilde{m} + K\widetilde{n})$ memory to get $\Delta\mathbf{S}[:, Q]$.*

$|\mathcal{D}_{out}|$ is the number of nodes satisfying the conditions of case 3. For a deleted node d, if the in-degree of some of node d's out-neighbours $\mathcal{O}(d)$ are larger than 1, then we define these out-neighbours as $\mathcal{D}_{out}$. Mathematically, $(\forall deg_a^- > 1 | a \in \mathcal{O}(d), a \in \mathcal{D}_{out})$.

time $O(K\widetilde{m})$ comes from the matrix-vector products (Eq. 3.29 and Eq. 3.30) $\tilde{\mathbf{A}}^T\mathbf{p}^{(k-1)}$, $\tilde{\mathbf{A}}^T\mathbf{t}^{(k-1)}$, $\tilde{\mathbf{A}}^T\mathbf{w}^{(k-1)}$, and $\tilde{\mathbf{A}}^T\mathbf{r}^{(k-1)}$. For each deleted node, the similarity scores are calculated by Eq. 3.29 and Eq. 3.30 in response to query Q; thus, the time complexity is $O(K(\widetilde{m} + |Q||\mathcal{D}_{out}|\widetilde{n}))$.

Similarity search algorithms over decremental dynamic graphs (node deletion) have been introduced separately in three cases. Now we take a network as an example, which includes all the 3 cases, to illustrate how the D-deCoSim (Node) algorithm works over a dynamic graph with all three cases and demonstrate the reliability of the similarity search algorithms.

Example 10. *In Figure 3.9, given an old graph with seven nodes such that the size of old adjacency matrix $\mathbf{A}$ is 7 and nodes e, f, g have been deleted from the old graph, $\Delta G = \Delta G_g \cup \Delta G_f \cup \Delta G_e$; then, given a query $q = a$, decay factory $C = 0.6$, and iteration number $K = 3$, the update similarity search scores can be generated using the algorithm as follows.*

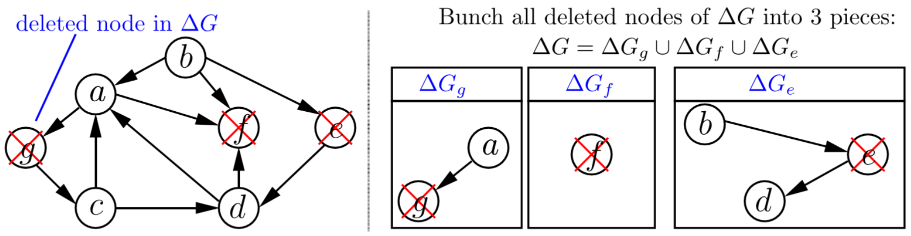

deleted node in ΔG
a
b
c
d
e
f
g
Bunch all deleted nodes of ΔG into 3 pieces:
$\Delta G = \Delta G_g \cup \Delta G_f \cup \Delta G_e$
ΔG_g
ΔG_f
ΔG_e

Figure 3.9: 4 Cases: Node Deletion ΔG of Graph G

Firstly, to calculate the updated similarity search score $\Delta\mathbf{S}_g[:, a]$ with respect to ΔG_g, since the out-neighbour of the deleted node g is node c and as it only has one in-neighbour node g, Theorem 8 is used to generate $\Delta\mathbf{S}_g[:, a]$.

1. *According to Eq.*(3.26)*, we have* $\mathbf{u} = [\,0\ 0\ 1\ 0\ 0\ 0\,]^T$*, based on the value of* $\mathbf{u}$*,* $\{\mathbf{p}^{(k)}\}$ *and* $\{\mathbf{t}^{(k)}\}$ *can be iteratively generated as*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0,0,1,0,0,0]^T$	$[-0.0185,0,-0.5357,-0.0082,0,-0.0222]^T$
1	$[0.3333,0,0,0.5,0,0]^T$	$[-0.1813,0,0,-0.2678,0,-0.0089]^T$
2	$[0.1667,0,0,0,0,0.2778]^T$	$[-0.0893,0,0,0,0,-0.1487]^T$
3	$[0,0,0,0,0,0.0556]^T$	$[0,0,0,0,0,-0.0298]^T$

2. *We use* $\Delta\mathbf{S}[:,a] = \sum_{k=0}^{3} C^k(\mathbf{p}^{(k)}[a](\mathbf{t}^{(k)})^T + \mathbf{t}^{(k)}[a](\mathbf{p}^{(k)})^T)$ *to derive* $\Delta\mathbf{S}_g[:,a]$ *as*

$$\Delta\mathbf{S}_g[:,a] = [-0.0832,0,-0.0185,-0.1079,0,-0.0197]^T$$

3. *After the CoSimRank score is updated with respect to* ΔG_g*, the graph changes from* G *to* $G_1 = G - \Delta G_g$*, and* $\mathbf{A} = \widetilde{\mathbf{A}}$*.*

Next, there are 6 nodes in the new graph G_1*, and node* f *is deleted from graph* G_1*. Since the out-degree of node* f *in the old graph* G_1 *is 0, according to Theorem 6, no computation is required, and the new similarity search score matrix* $\widetilde{\mathbf{S}}_f[:,a]$ *of the new graph* $\widetilde{G}$ *is equal to* $\Delta\mathbf{S}_{11}[:,a]$ *of the old graph, i.e.,* $\Delta\mathbf{S}_f[:,a] = [0,0,0,0,0]^T$*.*

Then, the graph is updated from G_1 *to* $G_2 = G_1 - \Delta G_f$*, and there are 5 nodes in the new graph. The node e will be deleted next. The out-neighbour of the node e in graph* G_2 *is d, and the in-degree of the node d is 2. Here, node d can be defined as* $\mathcal{D}_{out}$*.*

1. *The new adjacency matrix can be generated as follows:*

$$\begin{aligned}\widetilde{\mathbf{A}}(:,\mathcal{D}_{out}) &= \mathbf{A}_{11}(:,\mathcal{D}_{out}) + \frac{1}{\deg^-(\mathcal{D}_{out})(\deg^-(\mathcal{D}_{out})-1)}\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}} \\ &= \begin{bmatrix} 0 \\ 0 \\ \frac{1}{2} \\ 0 \end{bmatrix} + \frac{1}{1*2}\begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}\end{aligned}$$

2. *The update of the CoSimRank score with respect to* ΔG_e *is calculated by applying Theorem 10. According to Eq.*(3.29) *and Eq.*(3.30)*, we have* $\mathbf{u} = [\,0\ 0\ 0\ 1\,]^T$*.* $\{\mathbf{p}^{(k)}\}$*,* $\{\mathbf{t}^{(k)}\}$*,*

$\{\mathbf{r}^{(k)}\}$, *and* $\{\mathbf{w}^{(k)}\}$ *can be iteratively generated as*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 0, 0, 1]^T$	$[0.1, 0, 0, 0.2250]^T$
1	$[0.333, 0, 0, 0]^T$	$[0.075, 0, 0, 0]^T$
2	$[0, 0, 0, 0]^T$	$[0, 0, 0, 0]^T$
k	$\mathbf{r}^{(k)}$	$\mathbf{w}^{(k)}$
0	$[0, 0, 0, -0.5]^T$	$[0, 0, 0, 0.24]^T$
1	$[-0.1667, 0, 0, 0]^T$	$[0.08, 0, 0, 0]^T$
2	$[0, 0, 0, 0]^T$	$[0, 0, 0, 0]^T$

3. *We use* $\Delta\mathbf{S}[:, a] = \sum_{k=0}^{\infty} C^k(\mathbf{p}^{(k)}[a](\mathbf{t}^{(k)}) + \mathbf{t}^{(k)}[a](\mathbf{p}^{(k)}) + \mathbf{w}^{(k)}[a](\mathbf{r}^{(k)}) + \mathbf{r}^{(k)}[a](\mathbf{w}^{(k)}))$ *to derive* $\Delta\mathbf{S}_e[:, a]$ *as:* $\Delta\mathbf{S}_e[:, a] = [0.014, 0, 0, 0.1]^T$.

Finally, after deriving the CoSimRank scores in response to each update, we obtain the final $\Delta\mathbf{S}$ *of all updates. To keep the unit size of each update, here we define* $\mathcal{R}$ *as the remaining node set of new graph* $\widetilde{G}$*; in this example,* $\mathcal{R} = [a, b, c, d]$.

We can get the new CoSimRank scores of the new graph $\widetilde{G}$ *as* $\widetilde{\mathbf{S}}[\mathcal{R}, a] = \Delta\mathbf{S}_g[\mathcal{R}, a] + \Delta\mathbf{S}_f[\mathcal{R}, a] + \Delta\mathbf{S}_e[\mathcal{R}, a] + \mathbf{S}_{11}[\mathcal{R}, a] = [1.24, 0, 0, 0, 2]^T$.

Example 10 illustrates how the similarity search algorithm works on a dynamic graph with node deletion (incorporating all the three cases). Based on the computation of $\Delta\mathbf{S}_g$ (Case 2), $\Delta\mathbf{S}_f$ (Case 1), and $\Delta\mathbf{S}_e$ (Case 3), we found that the formula for Case 3 can be used to cover Case 1 and Case 2, but the computation corresponding to Case 1 and Case 2 are streamlined. Thus, the similarity search algorithm over a dynamic graph with node deletion can be attributed to Theorem 10.

3.3.3 Combinatorial Algorithms For Similarity Search Over Dynamic Graphs

Extracting similarity information in big data as efficiently as possible is an challenge in the era of information. In many situations, entities of big data and the links between data can be naturally represented in graph structures, such as: biocomputing (protein-protein interactions) (Du, 2010), recommendation systems(rating relations between customers and products) (Machado et al., 2014), social networks(users-users interaction) (Machado et al., 2014), etc. Therefore, efficient similarity search algorithms on graph structures are necessary and urgently needed.

Graphs are classified as static or dynamic based on whether or not the nodes and edges in the graph structure change. This subsection focus on similarity search algorithms on dynamic graphs.

According to Definition 2, dynamic graphs can be divided into incremental(nodes & edges addition) and decremental(nodes & edges deletion) dynamic graphs. Based on this, we proposed two efficient similarity search algorithms D-CoSim and D-deCoSim(D-deCoSim(Node) & D-deCoSim(Edge)) on incremental and decremental dynamic graphs respectively (Section 3.3.1 and Section 3.3.2). This section focuses on how to integrate the proposed techniques to perform similarity search on dynamic graphs efficiently.

In real life, the application of a combination of incremental and decremental dynamic graphs is more common, take Facebook social network as an example, as new users sign up or existed users log out, the number of users in the network fluctuates, as does the number of relationship edges in the network. To achieve efficient similarity search on dynamic graphs, we can combine D-CoSim and D-deCoSim,named Com-D.

Com-D is a combinatorial algorithm which includes D-CoSim and D-deCoSim(D-deCoSim(Node) & D-deCoSim(Edge)). These algorithms are independent of each other and can be called individually or in combination depending on the type of dynamic graph. It is worth noting that when the dynamic diagram contains the four cases in Figure 3.1, the order in which the individual algorithms are called directly affects the computational complexity and efficiency of the Com-D algorithm. The order in which the algorithms are called is D-deCoSim(Node), D-deCoSim(Edge) and D-CoSim. There are two advantages to calling the algorithms in this way. Calling D-deCoSim(Node) first not only reduces the size of the adjacency matrix directly, but also reduces the complexity of Com-D algorithm.

Example 11. *In Figure 3.10, given an old graph with five nodes such that the size of old adjacency matrix* $\mathbf{A}$ *is 5. Nodes e have been deleted from the old graph, edge* $(a \to b)$ *has been deleted and a new node* f *has been updated to graph* G*, thus* $\Delta G = \Delta G_e \cup \Delta G_b \cup \Delta G_f$*; then, given a query* $q = d$*, decay factory* $C = 0.6$*, and iteration number* $K = 3$*, the update similarity search scores can be generated using the algorithm as follows.*

To make Com-D algorithm more efficient, we first call D-deCoSim(Node) algorithm to update $\Delta\mathbf{S}_e[:, d]$ *with respect to the updated part* ΔG_e*, then D-deCoSim(Edge) algorithm will be called to generate* $\Delta\mathbf{S}_b[:, d]$ *with respect to the updated part* ΔG_b*, lastly, D-CoSim algorithm will be called*

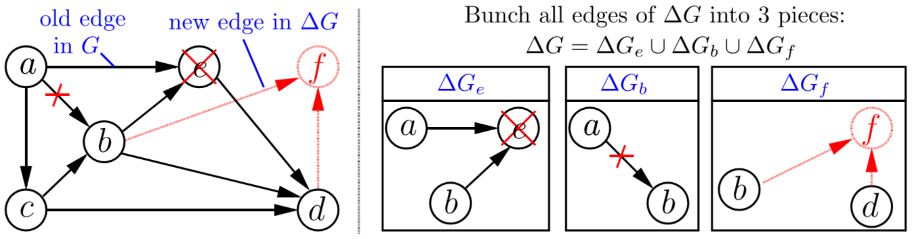

old edge in G
new edge in ΔG
Bunch all edges of ΔG into 3 pieces:
$\Delta G = \Delta G_e \cup \Delta G_b \cup \Delta G_f$
ΔG_e
ΔG_b
ΔG_f

Figure 3.10: Example of old web graph G(solid arrows) updated by ΔG

to calculate $\Delta\mathbf{S}_f[:,d]$ with respect to the updated part ΔG_f. The detailed steps of the calculation are shown below.

Firstly, to calculate the updated similarity search score $\Delta\mathbf{S}_e[:,d]$ with respect to ΔG_e. The out-neighbour of deleted node e is node d in graph G, and the in-degree of the node d is 2. Here, node d can be defined as $\mathcal{D}_{out}$.

1. *The new adjacency matrix can be generated as follows:*

$$\widetilde{\mathbf{A}}(:,\mathcal{D}_{out}) = \mathbf{A}_{11}(:,\mathcal{D}_{out}) + \frac{1}{\deg^-(\mathcal{D}_{out})(\deg^-(\mathcal{D}_{out})-1)}\mathbf{1}_{\{Nbr^-(\mathcal{D}_{out})\}}$$
$$= \begin{bmatrix} 0 \\ \frac{1}{3} \\ \frac{1}{3} \\ 0 \end{bmatrix} + \frac{1}{2*3}\begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 0 \\ \frac{1}{2} \\ \frac{1}{2} \\ 0 \end{bmatrix}$$

2. *The update of the CoSimRank score with respect to ΔG_e is calculated by applying Theorem 10. According to Eq.(3.29) and Eq.(3.30), we have $\mathbf{u} = [\,0\;0\;0\;\frac{1}{3}\,]^T$. $\{\mathbf{p}^{(k)}\}$, $\{\mathbf{t}^{(k)}\}$, $\{\mathbf{r}^{(k)}\}$, and $\{\mathbf{w}^{(k)}\}$ can be iteratively generated as*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0,0,0,-\frac{1}{3}]^T$	$[0,0.09,0,0.2349]^T$
1	$[0,0,0,0]^T$	$[0,0,0,0.045]^T$
2	$[0,0,0,0]^T$	$[0,0,0,0]^T$
3	$[0,0,0,0]^T$	$[0,0,0,0]^T$
k	$\mathbf{r}^{(k)}$	$\mathbf{w}^{(k)}$
0	$[0,0,0,1]^T$	$[0,0.095,0,0.1496]^T$
1	$[0,0,0,0]^T$	$[0,0,0,0.0475]^T$
2	$[0,0,0,0]^T$	$[0,0,0,0]^T$
3	$[0,0,0,0]^T$	$[0,0,0,0]^T$

3. *We use $\Delta\mathbf{S}[:,d] = \sum_{k=0}^{\infty} C^k(\mathbf{p}^{(k)}[d](\mathbf{t}^{(k)}) + \mathbf{t}^{(k)}[d](\mathbf{p}^{(k)}) + \mathbf{w}^{(k)}[d](\mathbf{r}^{(k)}) + \mathbf{r}^{(k)}[d](\mathbf{w}^{(k)}))$ to derive $\Delta\mathbf{S}_e[:,d]$ as: $\Delta\mathbf{S}_e[:,d] = [0,0.065,0,0.1426]^T$.*

Secondly, the graph is updated from G to $G_1 = G \ominus \Delta G_e$, and there are 4 nodes in the new graph. The edge $(a \to b)$ will be deleted next, and we generate $\Delta\mathbf{S}[:,d]$ with respect to ΔG_b. Because the in-degree of node b is not equal to 0 in the new graph G_1, according to Eq.(3.23) and Eq.(3.24), we set $\mathbf{w}^{(0)} = \delta_b \mathbf{A}[:,b] - \mathbf{1}_{\{v_1,\cdots,v_{\delta_b}\}}$ and $\mathbf{t}^{(0)} = \frac{C}{2(\delta_b - \deg_b^-)}(\mathbf{A} + \tilde{\mathbf{A}})^T \mathbf{r}$ to generate $\Delta\mathbf{S}[:,d]$ as follows:

1. *We compute $\mathbf{w}^{(k)}$ and $\mathbf{r}^{(k)}$ using Eq.(3.24):*

k	$\mathbf{w}^{(k)}$	$\mathbf{r}^{(k)}$
0	$[-0.5, 0, 0.5, 0]^T$	$[0, 0, 0, 0]^T$
1	$[0.5, 0, 0, 0]^T$	$[0, 0, 0, 0]^T$
2	$[0, 0, 0, 0]^T$	$[0.5, 0, 0, 0]^T$
3	$[0, 0, 0, 0]^T$	$[-0.5, 0.15, 0.8, 0]^T$

2. *Then, we obtain $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ through Eq.(3.23) with $\mathbf{r} = \mathbf{r}^{(4)}$:*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 1, 0, 0]^T$	$[0, 0.285, -0.3, 0.285]^T$
1	$[0, 0, 0, 0.5]^T$	$[0, -0.3, 0, 0, -0.0075]^T$
2	$[0, 0, 0, 0]^T$	$[0, 0, 0, -0.15]^T$
3	$[0, 0, 0, 0]^T$	$[0, 0, 0, 0]^T$

3. *After obtaining the values of $\{\mathbf{p}^{(3)}\}$ and $\{\mathbf{t}^{(3)}\}$, we derive $\Delta\mathbf{S}_b(:,d)$ using Eq.(3.22) in response to ΔG_b:*

$$\mathbf{\Delta S}_b[:,d] = \textstyle\sum_{k=0}^{3} 0.6^k \left(\mathbf{p}^{(k)}[d]\mathbf{t}^{(k)} + \mathbf{t}^{(k)}[d]\mathbf{p}^{(k)}\right)$$
$$= [0, 0.195, 0, -0.0045]^T \quad \square$$

Leveraging the updated CoSimRank values with respect to ΔG_b, the old graph is updated by $G_2 = G_1 \ominus \Delta G_b$, and let $\mathbf{A} = \widetilde{\mathbf{A}}$.

Finally, we generate ΔS_f with respect to ΔG_f. According to Theorem 4, we generate CoSimRank scores of graph G_2 as follows.

1. *First, we compute* $\{\mathbf{w}^{(k)}\}$ *and* $\{\mathbf{r}^{(k)}\}$ *using Eqs.*(3.10) *and* (3.9)*:*

k	$\mathbf{w}^{(k)}$	$\mathbf{r}^{(k)}$
0	$[0,1,0,1,0]^T$	$[0.5,0,0,0,0]^T$
1	$[0,0.5,1.5,0,0]^T$	$[1.5,0,0.8,0,0]^T$
2	$[1.5,0,0.5,0,0]^T$	$[0,0.98,2.4,0.24,0]^T$
3	$[0.5,0,0,0,0]^T$	$[0,2.44,0,2.014,0]^T$

2. *Next, we obtain* $\{\mathbf{p}^{(k)}\}$ *and* $\{\mathbf{t}^{(k)}\}$ *via Eq.*(3.8) *with* $\mathbf{r} = \mathbf{r}^{(3)}$*:*

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0,0,0,0,1]^T$	$[0,0,0,0.366,0.3341]^T$
1	$[0,0,0,0,0]^T$	$[0,0,0,0,0.183]^T$
2	$[0,0,0,0,0]^T$	$[0,0,0,0,0]^T$
3	$[0,0,0,0,0]^T$	$[0,0,0,0,0]^T$

3. *Finally, we use Eq.*(3.7) *to derive* $\mathbf{\Delta S_f}[:,d]$ *in response to* ΔG_f*:*

$$\begin{aligned}\mathbf{\Delta S_f}[:,d] &= \sum_{k=0}^{3} 0.6^k \left(\mathbf{t}^{(k)}[d]\cdot\mathbf{p}^{(k)} + \mathbf{p}^{(k)}[d]\cdot\mathbf{t}^{(k)}\right) \\ &= [0,0,0,0,0.366]^T\end{aligned}$$

□

The final CoSimRank scores of new graph G_2 *can be calculated by:*

$$\mathbf{S}_{G_2}[:,d] = \mathbf{S}_G[:,d] + \Delta\mathbf{S}_e[:,d] + \Delta\mathbf{S}_b[:,d] + \Delta\mathbf{S}_f[:,d] = [0,0.96,0,2.92,0.32]^T$$

3.4 Experimental Evaluation

This section presents an experimental study on real large-scale data to compare our algorithms with other baseline algorithms. Our evaluations using various datasets verify the superiority of D-CoSim and D-deCoSim with dynamic graphs.

The performance efficiency is evaluated using three metrics:

(a) Running Time. On incremental dynamic graphs, D-CoSim quickly responds to the CoSim-Rank search, with no need to recompute scores from scratch. On a decremental dynamic graph, D-deCoSim is much faster than the baseline approaches.

(b) Memory Space. Both D-CoSim and D-deCoSim require only linear memory, and they scale well on million-node graphs.

(c) Accuracy. D-CoSim and D-deCoSim do not compromise accuracy while improving (search) speed.

3.4.1 Experimental Setting

Datasets. We adopt the following public datasets:

Datasets		**#-Nodes**	**#-Edges**	**Type**
as-735	(AS)	7,716	26,467	Undirected
ca-HepPh	(HP)	12,008	237,010	Undirected
email-EuAll	(EE)	265,214	420,045	Directed
web-Google	(WG)	916,428	5,105,039	Directed
wiki-Talk	(WT)	2,394,385	5,021,410	Directed
soc-LiveJournal	(LJ)	18,520,486	298,113,762	Directed

Table 3.2: Description of Datasets

- as-735 (AS). It is a communication graph representing autonomous systems, taken from the Border Gateway Protocol logs, where an edge represent a who-talks-to-whom relationship.

- ca-HepPh (HP). It is a collaboration graph obtained from the arXiv High Energy Physics. If two authors (nodes) co-authored a paper, there is an edge between them.

- email-EuAll (EE). It is an EU email contact graph. Each node comprises an email address. If node i sent at least one message to j, there is an edge $i \to j$ in the network.

- web-Google (WG). It is a Google web graph, where each node is a web page and each edge a hyperlink.

- wiki-Talk (WT). On Wikipedia, each user (node) has a talk page that other users can edit for discussion. In this graph, an edge $i \to j$ means that user i edited user j's talk page.
- soc-LiveJournal (LJ). It is a social community network, where edge $i \to j$ is a friendship link from user i to j.

The size of each dataset has been illustrated in Table 3.2. To simulate real evolution on dynamic graphs, we used a random typing generator (RTG) (Akoglu & Faloutsos, 2009) to generate $|\Delta G|$ dynamic updates following linkage generation models (Leskovec, Kleinberg, & Faloutsos, 2007; Ntoulas, Cho, & Olston, 2004). Some graph operations have been done with Standard Network Analysis Platform (SNAP), and the details of SNAP is introduced in Appendix B.

All experiments have been conducted on a PC with Intel Core i7-6700 3.40GHz CPU and 64GB memory compiled by VC++.

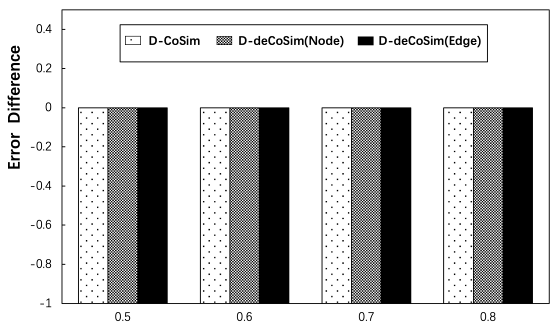
Compared Algorithms. We ran our D-CoSim and D-deCoSim over an incremental graph and a decremental dynamic graph respectively and compared them with two state-of-the-art CoSimRank competitors:

(a) CSR, a method by (Rothe & Schütze, 2014) that retrieves a CoSimRank score from the sum of the dot product of two personalised PageRank vectors;

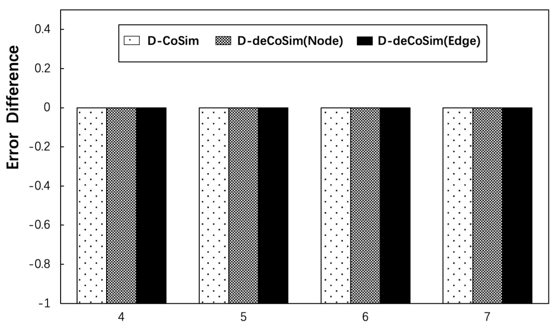
(b) CSM, a repeated-squaring method by (Yu & McCann, 2015a) that cuts down the number of CoSimRank iterations.

3.4.2 Experimental results

This section demonstrates the efficiency and accuracy of our similarity search algorithms by implementing them on different large-scale realistic graphs. Before testing the efficiency of the algorithm, we do some tests on the parameters (C and K) to ensure that the efficiency experiments are more accurate afterwards. We implement three algorithms, D-CoSim, D-deCoSim(Node) and D-deCoSim(Edge), on dataset EE with 1000 updates, respectively. Figure 3.11(a) shows that there are no error difference between the results of CSR and the three proposed algorithms with vary C. Figure 3.11(b) shows similarly trend as Figure 3.11(a), all error difference between the similarity scores of CSR and the three proposed algorithms with vary K are 0. Figure 3.11 illustrates that the proposed algorithms do not sacrifice any accuracy, so the parameter settings do not affect the efficiency of the proposed algorithm. Therefore the parameters in this chapter



(a) Vary C for Algorithms



(b) Vary K for Algorithms

Figure 3.11: Parameter Testings

are set as $C = 0.6, K = 5$, which are previously used in (Rothe & Schütze, 2014). After the settings of parameters, the experimental results of the D-CoSim algorithm over an incremental dynamic graph has been shown as follows.

3.4.2.1 Experimental Results of D-CoSim For Incremental Dynamic Graphs

We implement the D-CoSim algorithm and baseline algorithms over six practical datasets, and the performance efficiency is evaluated using three metrics: time efficiency, memory complexity, and accuracy.

Time Efficiency. We evaluate D-CoSim algorithm's time efficiency over incremental dynamic graphs.

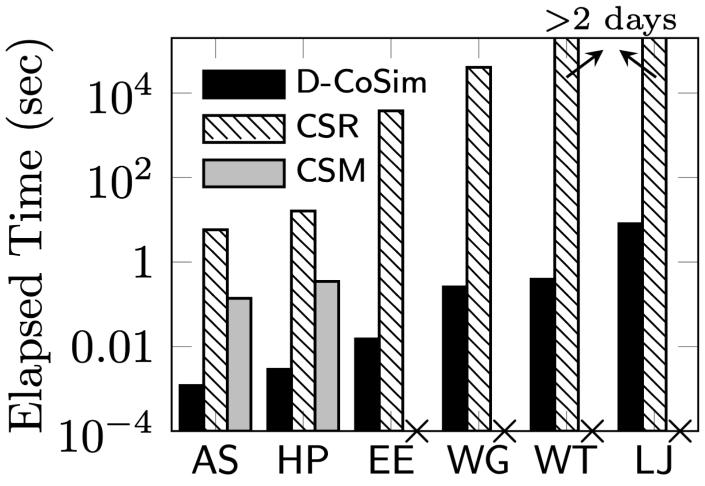


Figure 3.12: Time Efficiency on Incremental Graphs

Figure 3.12 depicts the time efficiency of D-CoSim on several dynamic graphs. From each dataset, we randomly select $|Q| = 500$ queries and build $|\Delta G| = 1000$ new edge updates.

Figure 3.12 compares the time of D-CoSim against CSR and CSM to compute CoSimRank changes per update for each query. We see that D-CoSim is consistently 3-5 order-of-magnitude faster than CSR (*resp.*118x faster than CSM). This is because D-CoSim employs Theorem 1, which evaluates only the "refreshed areas" of CoSimRank scores in response to graph updates, without recomputing all scores from scratch. Moreover, unlike CSM that crashes on large datasets (*e.g.,* WT, LJ) due to insufficient memory for repeated squaring memoisation, D-CoSim can update the similarity scores within one second.

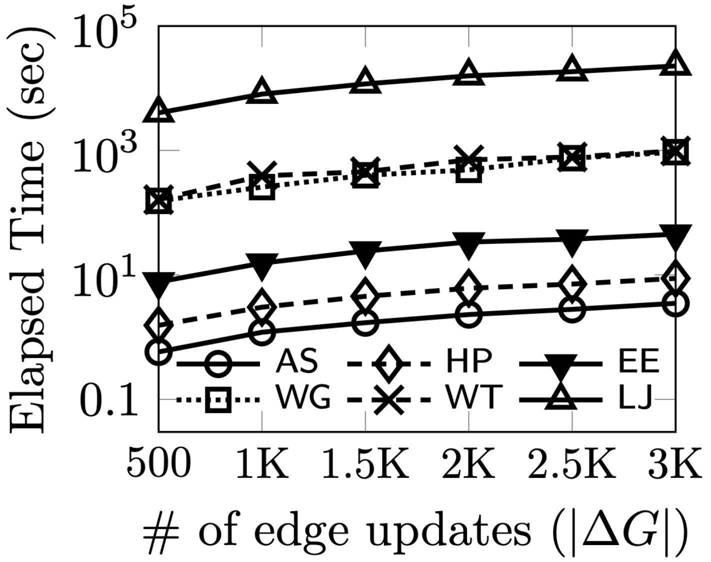


Figure 3.13: Time Efficiency on Incremental Graphs: Varied $|\Delta G|$ for D-CoSim

Figure 3.13 further depicts the time taken by D-CoSim with respect to $|\Delta G|$. As $|\Delta G|$ increases from 500 to 3000 in each dataset, the time of D-CoSim increases slightly, highlighting its scalability with respect to the number of edge updates. It is consistent with the time complexity in Theorem 4 where D-CoSim is linear to the number of update pieces p ($\leq \delta$).

Memory Space. We evaluate the memory efficiency and scalability of the D-CoSim algorithm over incremental graphs. Figure 3.14 depicts the memory efficiency of D-CoSim on six real datasets as compared with CSR and CSM. From each dataset, we randomly select $|Q| = 500$ queries. We generate new edge updates $|\Delta G| = 1000$ in each dataset for dynamic graphs. Figure 3.14 illustrates the memory of D-CoSim for ΔG updates in each dataset with respect to the query set Q. We see that D-CoSim and CSR have comparable memory; both increase linearly with the growing size of graphs, highlighting their scalability. On small datasets (*e.g.,* ASand HP) when CSM does not fail, the memory of D-CoSim is almost 2.5 orders of magnitude smaller

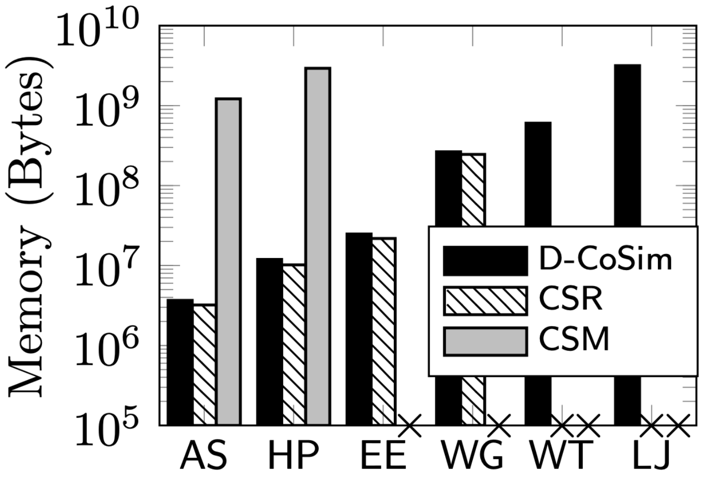


Figure 3.14: Memory Efficiency & Scalability of D-CoSim

than that of CSM. This is because D-CoSim only requires linear memory to store auxiliary vectors, as opposed to the $O(n^2)$ memory of CSM for repeated squaring. D-deCoSim exhibits similar memory efficiency with D-CoSim over decremental graphs.

Accuracy.

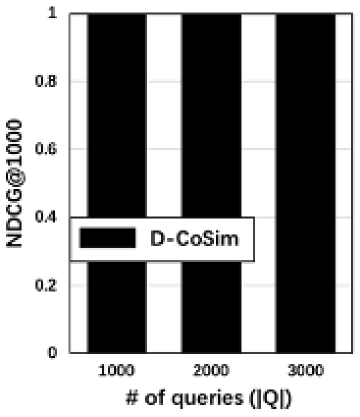


Figure 3.15: Accuracy of D-CoSim

We evaluate the accuracy of D-CoSim, relative to the original CSR, on real datasets. We randomly pick various query sets with size $|Q|$ varying from 1000 to 3000. For each query set Q, based on the CoSimRank scores $\mathbf{S}[:, Q]$ from D-CoSim, we measure their similarity ranking results using normalised discounted cumulative gain (NDCG) (Y. Wang et al., 2013):

$$\mathrm{NDCG}_Q@k = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \left(Z_{k,j} \sum_{x=1}^{k} \frac{2^{\mathbf{S}[x,q]} - 1}{\log_2(1+x)} \right)$$

where $Z_{k,j}$ is a normalization factor that is the DCG ranking results by the original method

CSR. Thus, NDCG = 1 implies that the CoSimRank ranking of the compared algorithm perfectly matches that of CSR, with no accuracy loss. Figure 3.15 shows the accuracy of D-CoSim using NDCG for top $k = 1000$ CoSimRank ranking scores on AS. The trends on other datasets are similar, so we omit them here. From the results, we notice that, for each query set Q, NDCGs of D-CoSim equal to 1s, implying that D-CoSim does not sacrifice accuracy for increased speed. This verifies the correctness of Theorem 1.

Next, we evaluate D-deCoSim similarity search algorithm over decremental dynamic graphs with edge deletion and node deletion separately; we define D-deCoSim (Edge) as the similarity search algorithm over decremental dynamic graphs with edge deletion and D-deCoSim (Node) as the algorithm over dynamic graphs with node deletion.

3.4.2.2 Experimental Evaluation of D-deCoSim Algorithm over Decremental Dynamic Graphs

As introduced in Section 3.3.2, the D-deCoSim algorithm can be separated into two parts: D-deCoSim (Edge) works on dynamic graphs with edge deletion, and D-deCoSim (Node) works efficiently over dynamic graphs with node deletion. This section evaluates the time efficiency, memory complexity, and accuracy of D-deCoSim (Edge) and D-deCoSim (Node) separately.

Time Efficiency. First, we implement D-deCoSim (Edge) over six realistic large-scale graphs to test its efficiency, scalability, and accuracy.

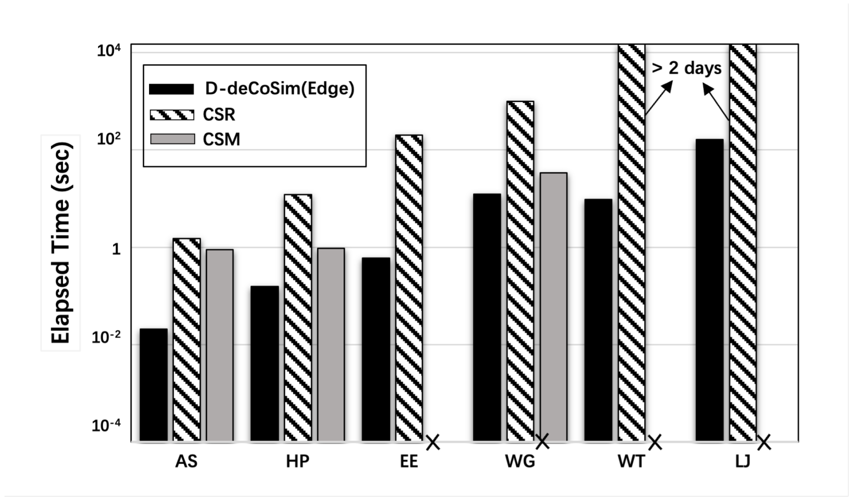


Figure 3.16: Time Efficiency on Decremental Graphs (Edge deletion)

Figure 3.16 depicts the time efficiency of the D-deCoSim algorithm (Edge Deletion) over six different realistic dynamic networks (graphs), and the elapsed time comes from the computation of CoSimRank changes per update for each query. From each dataset, we randomly take $|Q| = 500$ queries and remove edges $|\Delta G| = 1000$ from the graph. On the first two graphs (as-735 and ca-HepPh), the D-deCoSim(Edge) algorithm shows remarkable advantage over CSR and CSM algorithms; D-deCoSim is around 80 times faster than the CSR algorithm on the first two graphs and more than 60 times faster than the CSM algorithm. For the remaining 4 large-scale graphs (*e.g.,* email-EuAll, web-Google, wiki-Talk, and soc-LiveJournal), D-deCoSim exhibits a significant advantage over CSR; it is steadily 3-5 orders of magnitude faster than the CSR algorithm. The CSM algorithm cannot even be implemented over the four large-scale networks since the algorithm needs large memory space. However, the D-deCoSim (Edge) algorithm can update CoSimRank scores in 1 second (*e.g.,* EE) over large-scale networks, which indicates its scalability.

Next, D-deCoSim (Node) is implemented on the same six large graphs to evaluate the time efficiency of the algorithm.

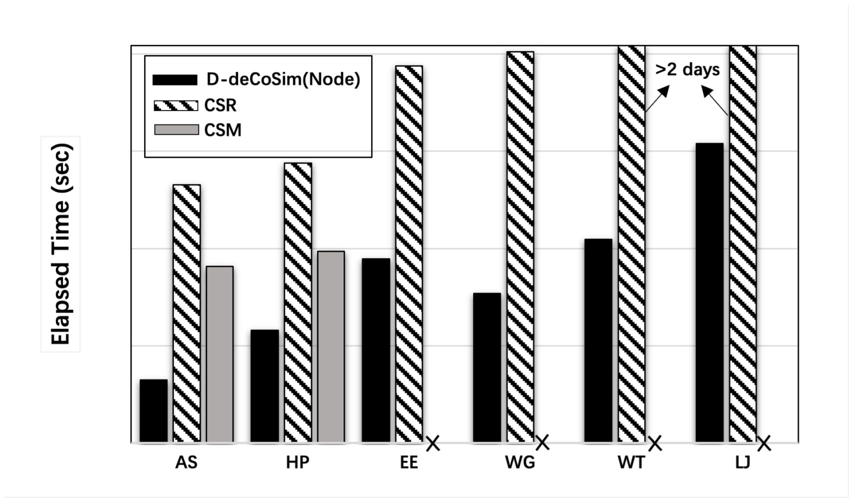


Figure 3.17: Time Efficiency on Decremental Graphs (Node Deletion)

Figure 3.17 illustrates the experimental result showing the time efficiency of D-deCoSim (Node) over decremental graphs. With the same settings provided in Figure 3.16, we take $|Q| = 500.$ and $|\Delta G| = 1000$. Note that for dynamic graphs with node deletion, the number of deleted nodes $\widetilde{n}$ cannot represent the number of update pieces of graph G since each deleted node may contain several different cases that depend on the out-degree of the deleted node and the in-degree of $\mathcal{D}_d$. Thus, we set $\widetilde{n} = 1000$ for each network, but the number of update pieces of each graph may be different; further details can be found in Figure 3.18. We can learn from Figure 3.17 that the D-deCoSim (Node) algorithm is consistently around 600x faster than the CSR algorithm over AS and HP graphs, and it is 305x faster than CSR on the email-EuAll graph. D-deCoSim (Node) is significantly efficient than CSR over large-scale networks,*e.g.,* WG, WT, and LJ, and it is constantly 3-5 orders of magnitude faster than the CSR algorithm. For the CSM algorithm, D-deCoSim (Node) is over 350x faster than the CSR algorithm on AS graph and 46x faster on HP graph. Furthermore, due to the memory space cost, CSM still cannot work over large-scale dynamic networks (*e.g.,* EE, WG, WT, and LJ) with node deletion. Comparing the trends of D-deCoSim (Edge) and D-deCoSim (Node) in Figure 3.16 (black bar) and Figure 3.17 (black bar), a similar growing trend with respect to to the size of graph is observed, and both are efficient algorithms that compute the CoSimRank scores over million-size graphs.

The D-deCoSim (Node) algorithm includes three cases; next, we evaluate the time efficiency of all three cases separately over large-scale graphs.

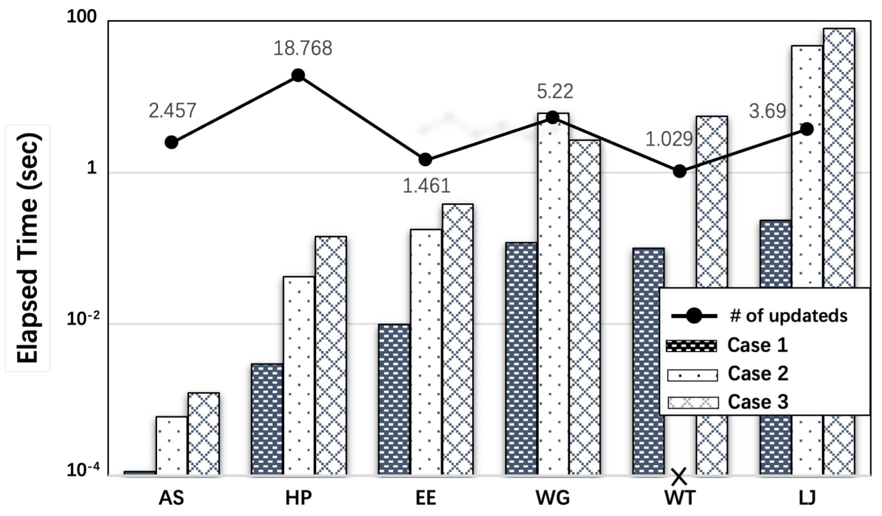


Figure 3.18: Time Efficiency of All 3 Cases in D-deCoSim (Node)

Figure 3.18 shows the time cost for the three cases (bar chart) and the number of update pieces (line chart). For the six graphs, the number of deleted nodes is $\widetilde{n} = 1000$ and query number is $|Q| = 500$. Since each deleted node may contain several different cases, the number of deleted nodes ($\widetilde{n}$) is not the number of updates $|\Delta G|$ of the graph. The line chart in Figure 3.18 indicates the number of updated pieces in the dynamic graph, and the unit of each value is 10^2. In the dynamic graph, each deleted node represent at least one update piece; thus, $\widetilde{n} \leq |\Delta G|$, and the values in line chart are larger than 1000, and the number of updated pieces is not increased by the growth of the size of the graph; it only relevant to the number of $\mathcal{D}_d$ and the in-degree of $\mathcal{D}_d$ ($deg^-_{\mathcal{D}_d}$). The bar chart in 3.18 compares the time for each case of D-deCoSim (Node) to compute CSR changes per update for each query. We see that the time for Case 1 over six different graphs is consistently faster than in Case 2 and Case 3 since the computation complexity for Case 1 is $O(\delta|Q|)$ (Theorem 7), which is affected by the number of deleted nodes and the size of the graph. The computation time for Case 2 and Case 3 has comparable cost, and the experimental results accord with Theorem 9 and Theorem 11. For the time complexity of three cases of the D-deCoSim (Node) algorithm, all of them are increasing mildly, highlighting the scalability of D-deCoSim. Note that the time cost for Case 2 over WT graph is 0, which means 1000 deleted nodes do not belong to Case 2, so the result is zero.

Then, we evaluate the memory efficiency of D-deCoSim (Edge) and D-deCoSim) (Node) respectively over large-scale detrimental graphs as compared with CSR and CSM.

Memory Efficiency. We test the memory efficiency of D-deCoSim (Node) on 6 real decremental dynamic graphs.

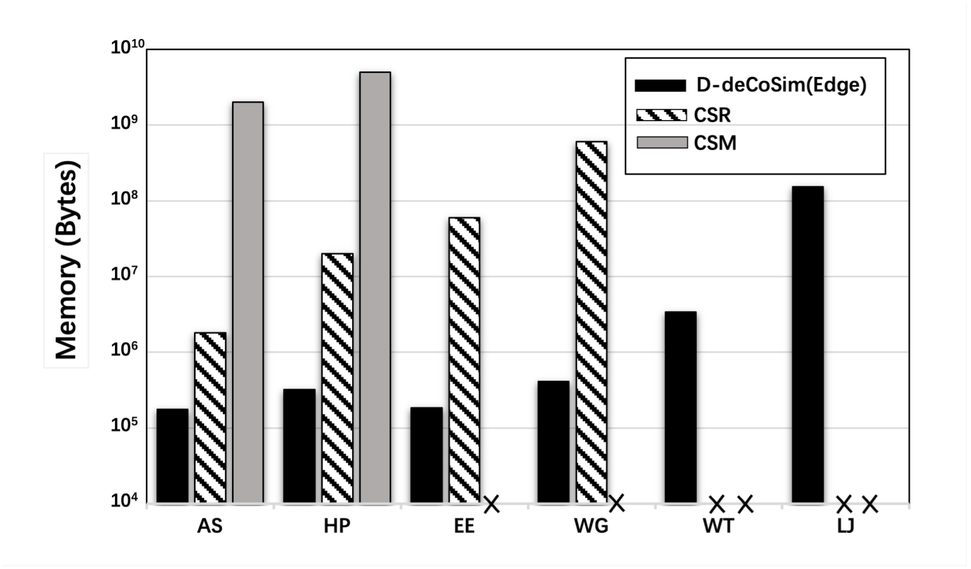


Figure 3.19: Memory Efficiency & Scalability of D-deCoSim(Edge)

Figure 3.19 shows the memory efficiency of the D-deCoSim (Edge) algorithm on six real datasets compared with CSR and CSM. We randomly selected 500 nodes in the query set Q, and 1000 deleted edges have been generated on each graph. We can learn from the figure that the memory efficiency of D-deCoSim (Edge) always shows a great advantage over CSR and CSM on the 6 graphs. On small datasets (*e.g.,* ASand HP), the memory complexity of CSM is over 4 orders of magnitude larger than D-deCoSim(edge). This is because of the memory complexity is linear to the number of updates; otherwise, the memory space required by CSM is $O(n^2)$, which keeps squaring; thus, CSM cannot be implemented over large-scale graphs. The memory efficiency of CSR is smaller than that of the CSM algorithm over smaller graphs (*e.g.,* AS and HP), but it is up to 4 orders of magnitude larger than D-deCoSim (Edge) algorithm over AS, HP, EE and WG. The memory complexity of D-deCoSim (Edge) guarantees efficiency and scalability.

Next, we implement D-deCoSim (Node), CSR and CSM on six realistic datasets to evaluate their memory efficiency.

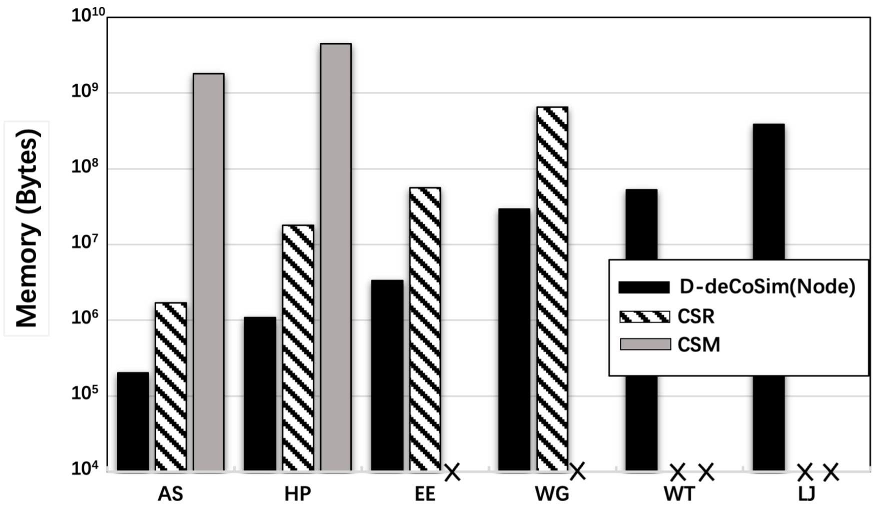


Figure 3.20: Memory Efficiency & Scalability of D-deCoSim(Node)

In Figure 3.20, we randomly took 500 nodes as query Q and deleted 1000 nodes from each graph; thus, $\widetilde{n} = 1000$ and the number of updates $|\Delta G| \leq 1000$. The CSM algorithm requires highest memory space cost over AS and HP due to its repeat-squaring memory space cost. The CSR algorithm consistently requires more memory space cost for updating the CoSimRank scores of decremental graphs (*e.g.,* AS, HP, EEand WG) than the D-deCoSim (Node) algorithm; the memory efficiency of D-deCoSim (Node) is up to 2 orders of magnitude smaller than that of the CSR algorithm. The black bar (D-deCoSim (Node)) is increasing slowly with the growing size of graphs, which guarantees the scalability of the D-deCoSim (Node) algorithm.

Finally, we evaluate the accuracy of D-deCoSim (Edge) and D-deCoSim (Node) respectively compared with the CoSimRank scores of CSR on real datasets.

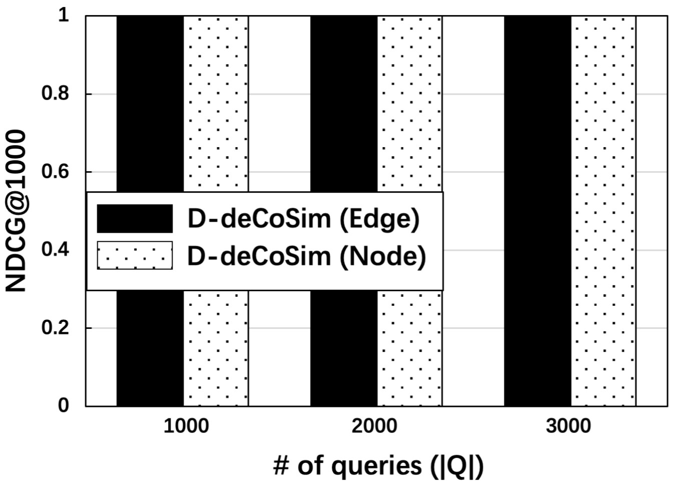


Figure 3.21: Accuracy of D-deCoSim

We randomly select different sizes of query, $|Q| = [1000, 2000, 3000]$. For each query set Q, we evaluate CoSimRank scores $\mathbf{S}[:, Q]$ of D-deCoSim (Edge and Node) using NDCG (normalised discounted cumulative gain) (Y. Wang et al., 2013). The measuring method has been introduced in Section 3.4.2.1. $NDCG = 1$ means that the algorithm perfectly matches the similarity search results of CSR without any accuracy loss. In Figure 3.21, the top $k = 1000$ CoSimRank scores of D-deCoSim (Edge) and D-deCoSim (Node) on graph AS have been measured using NDCG. The trends corresponding to other datasets are identical. Thus, these results are omitted here. We can learn from the figure that the NDCGs of D-deCoSim (Edge) and D-deCoSim (Node) are 1s, which means both do not sacrifice accuracy for increased speed. This proves the accuracy of Theorem 4 to Theorem 10.

3.5 Related Works

Similarity search over large-scale dynamic graphs is an exceptional case due to consistent changes (edges/nodes). The demand for similarity search algorithm over large-scale dynamic graphs is universal, which finds applications in recommend systems (Hang & Singh, 2010) (changes come from the number of items and the links between objects), co-citation networks (Eto, 2019), web search (Fogaras & Rácz, 2005), etc.

Different from the similarity search on dynamic graphs, many existing algorithms carry out similarity search over static graphs. Previous research on CoSimRank search focuses on static graphs. The pioneering research of (Rothe & Schütze, 2014) proposed an efficient local algorithm that computes each CoSimRank score from the sum of the dot product of two personalised PageRank vectors. It requires $O(Kdn)$ time and $O(dn)$ memory to compute a single pair CoSimRank score over a static graph with n nodes and d average degree after K iterations. However, when the graph is slightly updated, all CoSimRank scores must be recomputed from scratch. Recently, Yu and McCann (Yu & McCann, 2015a) have suggested an optimisation technique, namely CoSimMate, that leverages repeated squaring memoisation to cut down the number of iterations from K to $\lceil \log_2 K \rceil$ for all pairs of CoSimRank scores' retrieval. However, this approach requires an extra $O(n^2)$ memory to store repeated squaring results, which is impractical for large-scale graphs. Worse still, the approach of (Yu & McCann, 2015a) is a non-local algorithm on static graphs, meaning even if one wishes to compute a single-pair score, scores of all pairs must be computed simultaneously.

Regarding dynamic updating, no work on CoSimRank could be identified except for a relatively small work on the updating of SimRank, a variant of SimRank, in dynamic graphs (Yu, Lin, Zhang, & McCann, 2018b; Jiang, Fu, Wong, & Wang, 2017; Shao, Cui, Chen, Liu, & Xie, 2015b; Yu, Lin, & Zhang, 2014a; C. Li et al., 2010). However, when extended to CoSimRank, these works would be inadequate due to the following reasons: First, the two state-of-the-art studies (Jiang et al., 2017; Shao et al., 2015b) are based on random walk sampling, whose optimisation techniques heavily hinge on aggregating "only the *first* meeting time" of two random surfers for SimRank. If applied to aggregate "*all* the meeting times" of two random surfers for CoSimRank, their approaches will become slow due to the expense to sample additional meeting paths of two coalescing random walks. Second, some works (C. Li et al., 2010; Yu, Lin, & Zhang, 2014a) devised low-rank decomposition methods to update SimRank scores of all pairs, leading to $O(n^2)$ memory to store the decomposed matrices, which is not scalable for large graphs. Worse still, these methods rest on an assumption that *all pairs* of old SimRank scores should be given in advance even if only a few pairs of scores need updating, which is unrealistic in practice.

There has also been much research on computing incremental personalised PageRank (PPR) vectors (Bahmani, Chowdhury, & Goel, 2010), and dynamic Random Walk with Restart (RWR) proximities (Yu & McCann, 2016). However, directly applying these techniques in dynamic CoSimRank updating is not efficient. This is because the CoSimRank score at iteration k is the sum of k inner products between two personalised PageRank vectors at every iteration $i = 1, 2, \cdots, k$. Thus, to update the k-th iterative CoSimRank score, existing incremental PPR (RWR) algorithms will be repeatedly applied $2k$ times to update two PPR (RWR) vectors at every iteration $i = 1, 2, \cdots, k$, respectively, before summing up the k dot products of every two PPR (RWR) vectors at each iteration, which would be rather expensive. To bridge this gap, we proposed novel D-CoSim and D-deCoSim algorithms to efficiently and accurately update the similarity scores of dynamic graphs.

3.6 Conclusion

This section presents two dynamic schemes, D-CoSim and D-deCoSim, for fast, accurate CoSimRank retrieval from evolving graphs. For incremental dynamic graphs, we devise a novel approach D-CoSim that (a) bunches all edges of ΔG into pieces $\{\Delta G_i\}$ and (b) characterises only the CoSimRank changes in response to each update piece ΔG_i as the linear combination of vectors, thus discarding unnecessary computations by maximally sharing common intermediate information on each ΔG_i. For decremental dynamic graphs, we present an efficient and accurate approach D-deCoSim that can be separated into two parts: (a) D-deCoSim (edge) is a similarity search algorithm applied over decremental dynamic graphs with edge deletion. It bunches together all deleted edges ΔG into pieces $\{\Delta G_i\}$ firstly, and then it retrieves the CoSimRank scores of "refreshed area" only in response to the query, which omits the repeated computations. (b) D-deCoSim(Node) is a similarity search algorithm applied over decremental dynamic graphs with node deletion. For each deleted node, the algorithm bunched together out-neighbours of the deleted nodes into three cases based on the number of $\mathcal{D}_d$ and the in-degree of $\mathcal{D}_d$ ($\deg^-_{\mathcal{D}_d}$) firstly, and then it follows the theorem relevant to three cases to compute the CoSimRank scores in response to query Q respectively. The experimental results on real large-scale datasets have shown that D-CoSim and D-deCoSim outperform the best-known algorithm CSR and CSM by around 3-5 orders of magnitude (time efficiency), for memory efficiency, CSM is around 4 orders of magnitude larger than D-CoSim and D-deCoSim, and CSR has comparable memory complexity with D-CoSim; both of them increase linearly with the growing size of graphs. The D-deCoSim algorithm is 1 order of magnitude smaller than CSR. D-CoSim and D-deCoSim retrieve CoSimRank quickly and accurately from dynamic graphs, with no need to reassess them from scratch.

4 Scalable Similarity Search over Large-Scale Graphs

In the previous chapter (Chapter 3), we proposed two efficient and accurate similarity search algorithms, D-CoSim and D-deCoSim, over large-scale dynamic graphs. The D-CoSim algorithm, apart from supporting fast dynamic CoSimRank retrieval on evolving graphs, can also be implemented on static graphs for accelerating similarity search. In this chapter, we propose an efficient scheme based on D-CoSim, named F-CoSim, that can dramatically speed up the CoSimRank search over large-scale static graphs. Furthermore, we propose several optimisation operations to speed up the F-CoSim algorithm without losing any accuracy.

4.1 Introduction

In the last decade, with the development of digital technology, humans have stepped into the age of information. To efficiently organise incremental information (nodes) and relations (edges), displaying them in a graph structure is a feasible operation. Moreover, identifying similar objects on a graph is a fundamental operation for extracting information from a large-scale graph. Thus, there is a growing need to automatically, efficiently and effectively derive similarity scores over large-scale graphs.

Following are several examples of similarity search in real-world applications, motivating the need for efficient algorithms to handle such assessment over large-scale static graphs.

Application 1 (Location-based Online Social Networks). Location-based online social network is a location-based social networking website where users share their locations by checking in. The friendship network is undirected and collected using some public APIs. The location-based online social network consists of users' location information (nodes) and friendship relation between users (edges). Thus, a location-based online social network can not only express the location of users but also display the interaction between them. A similarity search over location-based online social networks plays an essential role in people's lives. Especially when the COVID-19 virus is spreading globally, similarity search plays a vital role in tracking patients, and forecasting and predicting infectious disease epidemics.

Generally, the size of a location-based online social network of a city/country is huge; thus, it is a tremendous challenge to perform a similarity retrieval over it efficiently. This chapter proposes an efficient and accurate similarity search algorithm over large-scale networks to solve this problem.

Application 2 (Road Networks). Building a road network graph model is a fundamental method to deploy information on local authority roads, motorways and trunk roads. It has become more and more critical due to rapidly increasing urbanisation and development of infrastructure. Intersections and endpoints are expressed as nodes, and roads connecting these intersections or endpoints are represented as edges. The community search over road networks has become increasingly important in several real-life applications such as urban/city planning (Rui, 2013; R. Zhang, Rong, Wu, & Zhuo, 2020; Xiangxue, Lunhui, & Kaixun, 2019), social studies on local communities (Cannistraci, Alanis-Lobato, & Ravasi, 2013), and community recommendations by real estate agencies (Liu & Wang, 2016). However, the existing similarity search algorithms that retrieve similarity scores over large-scale networks are not efficient enough. This chapter leverages a fast algorithm to retrieve similarity scores over large-scale graphs.

4.1.1 Motivation

As mentioned earlier, a similarity search on a graph structure is widely used in our daily life, for example, web mining (Zheng, Zou, Lian, Wang, & Zhao, 2014), research on protein structure (J. Kim, Choi, & Li, 2019), recommendation system (Catherine & Cohen, 2016), patient tracking (J. Wu, 2021), etc. Due to the wide application of similarity search algorithms, an algorithm that can accurately reflect the similarity scores between node pair of large-scale graphs is essential. At the same time, with the development of society, the data in actual applications will become more prominent. Therefore, a similarity detection algorithm that can accurately and efficiently process and analyse big data is urgently needed.

Apart from the inefficiency in handling evolving dynamic graphs (Chapter 3), previous works on similarity search based on topology structure have the following limitations.

Firstly, we found that some existing similarity detection algorithms face an issue called “zero issue”. “Zero issue” refers to the notion that similarity scores of some node pairs should not be zero, but due to the limitation settings of similarity search algorithms, the similarity scores of some node pairs are equal to 0. The fundamental similarity search algorithm, SimRank (Jeh & Widom, 2002), indeed faces this issue. In the SimRank algorithm, for any two nodes a and b ($(a, b) \in \mathbf{V}$), if there does not exist any symmetric in-link path of node pair (a, b), the similarity score of the node pair $s(a, b)$ is 0. Furthermore, even if the similarity score of the node pair (a, b) is not equal to 0, $s(a, b)$ may still partially miss all the information due to dis-symmetric in-link paths for node pair (a, b). Thus, similarity search algorithms with a “zero issue” would miss partial information of graphs. In this way, the similarity search scores of a graph that displays the graph’s information partially may lead to some wrong decisions. For example, when we use SimRank to perform a similarity search on location-based online social networks, the results of SimRank algorithm are used to track patients and circulate situation forecast. In this case, incomplete similarity scores may lead to wrong prediction. To address this issue, we propose an algorithm that generates CoSimRank scores of large-scale static graphs. Unlike the SimRank algorithm, which only captures the first meeting time of two random surfaces, CoSimRank can capture all meeting times of two random surfaces. CoSimRank scores are, therefore, more reliable than SimRank scores.

Secondly, the existing works on similarity search on graphs may also have computation efficiency issues. The pioneering research by Sascha Rothe et al. proposed an efficient local algorithm, CoSimRank (Rothe & Schütze, 2014), that computes each CoSimRank score from the sum of the dot product of two personalised pagerank vectors. As we mentioned before, a CoSimRank score can capture all the meeting times of two random surfaces, making the results of CoSimRank more reliable than SimRank. However, since the CoSimRank algorithm (Rothe & Schütze, 2014) needs to go through “all the meeting times” of two random surfaces, it takes more time to sample all the meeting paths of two coalescing random walks; this process makes the approach slower compared to SimRank. The CoSimRank algorithm entails $O(Kdn)$ time and $O(dn)$ memory to compute a single pair similarity score on a static graph with n nodes and d average degree after K iterations. In a real application, it is essential to retrieve similarity scores instantly. Consider, for example, a recommendation system, where a user wants to buy a product similar to the previous one. If the recommendation list takes a long time to show the results, it will lose its reference. This chapter proposes an efficient similarity search algorithm that can retrieve CoSimRank scores with respect to a query on million-edge graphs within a second.

Moreover, most existing works on similarity search devised low-rank decomposition methods to update all pairs of similarity scores that are not scalable on large-scale graphs. (Yu & McCann, 2015a) proposed a fast similarity search algorithm on graphs, called Co-Simmate. Co-Simmate can retrieve similarity scores on small graphs efficiently, but it needs $O(n^2)$ memory to store the decomposed matrices. However, since it requires repeated squaring memory, Co-Simmate cannot be implemented on large-scale graphs.

These limitations of existing works motivate us to propose an efficient, effective and accurate similarity search algorithm for large-scale graphs with small memory usage.

Problem (Calculate CoSimRank scores on Static Large Graphs).

Given: a graph G, and a query set $Q = \{q_1, q_2, \cdots\}$

Retrieve: the CoSimRank scores with respect to Q on G quickly and accurately.

To speed up the computation of CoSimRank scores $\mathbf{S}$ over the static graph G, (i) F-CoSim first needs to find a "spanning polytree" T over G; (ii) on the "spanning polytree" T, we devise a fast approach to compute the CoSimRank scores $\mathbf{S}(T)$ of T; (iii) on $(G \ominus T)$, we employ D-CoSim to compute the changes of $\mathbf{S}(T)$ with respect to the delta graph $(G \ominus T)$(Chapter 3). With these ideas, F-CoSim has the following salient features:

- **Fast:** F-CoSim is an order of a magnitude faster than the best-known competitors on static graphs.
- **Accurate:** F-CoSim does not compromise on accuracy for speed.
- **Scalable:** Our schemes require only linear memory space and scale well on million-node graphs.

In a nutshell, static F-CoSim allows for efficient and accurate handling a myriad of SimRank-based applications (Fogaras & Rácz, 2005; Zhao, Han, & Sun, 2009; Sun, Han, Yan, Yu, & Wu, 2011; Lin, Lyu, & King, 2006).

4.1.2 Chapter Outlines

In this chapter, my main contributions are listed as follow:

Section 4.2 In this section, we characterise the differences between spanning polytree and spanning tree. Then, we propose a schema to extract spanning polytrees from graphs.

Section 4.3 This section introduces a novel operation to produce CoSimRank scores of a spanning polytree. We compute similarity search scores of spanning polytrees level by level, owing to the special structure of spanning polytrees.

Section 4.4 Based on the CoSimRank scores of spanning polytrees, we propose a fast and accurate similarity search algorithm, F-CoSim, on large-scale networks. In this algorithm, we recall the D-CoSim algorithm to generate $\Delta\mathbf{S}$, which comes from the different areas of the spanning polytree and the original graph.

Section 4.5 In this section, we propose three different schemes to optimize F-CoSim on large-scale graphs.

Section 4.5.1 We implement the Heapsort process to speed up the operation of extracting spanning polytrees from the graphs. The optimised method of extracting spanning polytree can more efficiently extract spanning polytrees from the graphs with fewer levels.

Section 4.5.2 In this section, we leverage an efficient single-source CoSimRank score retrieval of spanning polytree. Since F-CoSim is a single-source similarity search algorithm, it can efficiently show the relationship of the query. To efficiently compute the similarity scores with respect to query q over graphs, we compute single-source CoSimRank scores of spanning polytrees that can significantly reduce redundant computations.

Section 4.5.3 In this section, we implement parallel computing to speed up the F-CoSim algorithm on large-scale graphs. Since the similarity search of each level of a spanning polytree is independent, it can be implemented by an independent processor simultaneously.

Section 4.6 In this section, we conduct extensive experiments on real and large datasets. Through these experiments, we demonstrate the following facts. (1) Our efficient similarity search algorithm, F-CoSim, outperforms state-of-the-art approaches on static graphs with a speed-up of up to 9.8 times. (2) F-CoSim shows great advantage on memory efficiency to CSM. (3) F-CoSim does not compromise on accuracy for speed. (4) The optimizations of F-CoSim, Opt_F-CoSim algorithm, is more efficient than F-CoSim algorithm.

4.2 Why Spanning Polytree is Irreplaceable

In this section, we introduce the notions of a spanning tree and spanning polytree. Furthermore, we illustrate the reason why only spanning polytrees can be implemented in our similarity search algorithm on large-scale static graphs.

The Differences Between Spanning Polytree and Spanning Tree.

Extracting a spanning tree from a graph is a fundamental operation method in the practical application of the graph theory. There are several vivid live examples such as cluster analysis (Y. Wang, Yu, Gu, & Shun, 2021; Kireyeu, 2021), water supply network design (Vegas Niño, Martínez Alzamora, & Tzatchkov, 2021), electrical grid system optimization (Gupta, Khodabakhsh, Mortagy, & Nikolova, 2021), civil network routing protocol (Z. Wang, Chen, & Li, 2021), etc.

Definition 5 (**Spanning Tree**). *A spanning tree T of a connected graph G is a subgraph of G that includes every vertex of G, covering it with the minimum possible number of edges. T, therefore, contains no cycles, and cannot be disconnected.*

According to Definition 5, for every connected and undirected graph G, at least one spanning tree T exists. On the contrary, no spanning tree exists in a disconnected graph, as all vertices cannot be spanned in a single subgraph.

The spanning tree T of a graph G has the following general properties (G1 - G7) (Gross & Tucker, 2009; Gross, Yellen, & Anderson, 2018):

G1 For every connected graph G, the number of spanning tree $N(T) \geq 1$.

G2 Any spanning tree T of graph G has the same number of vertices and edges.

G3 There is no loop in a spanning tree.

G4 Spanning tree T is minimally connected, as removing any one edge of T would cause the subgraph to be disconnected.

G5 Spanning tree T is maximal acyclic, as adding an edge to a spanning tree T would create a cycle (loop) in the subgraph.

G6 There is at least one spanning tree T in any connected and undirected graph.

G7 In a directed graph G, the minimum spanning tree T has minimum edge weight.

A spanning tree has the following mathematical properties (M1 - M3):

M1 The number of the edge of a spanning tree T is $(n-1)$, where n is the number of vertices of graph G.

M2 In a complete graph G, if maximum $e-n+1$ edges from the graph are removed, then a spanning tree can be constructed, where e is the number of edges, and n is the number of nodes on the graph.

M3 In a complete graph G, there are a maximum of $n^{(n-2)}$ spanning trees.

These essential properties are what make spanning trees widely applicable in practical applications (*e.g.,* network redundancy, telecommunication networks). However, spanning trees have a significant limitation when applied in a topological structure. When an undirected graph is converted to a directed graph, there is no guarantee that the spanning tree will exist. The algorithm (similarity search over large-scale static graphs) presented later in this chapter 4.4 needs to cover directed graphs. The similarity search algorithm involving the extraction of spanning trees will lose its generality if the existence of a spanning tree in all directed graphs is not guaranteed. This limitation of spanning trees will prevent them from being widely used in practical directed networks. To overcome this limitation, a spanning polytree is generated for the similarity search algorithm.

Definition 6 (**Spanning Polytree**). *A spanning polytree T of a connected graph G is a subgraph of G that includes every node of G, with a maximal set of edges of G that contains no undirected cycles if we replace all the directed edges of T with undirected edges.*

Intuitively, in contrast with the traditional definition of a spanning tree, in which each node has only one parent node, our spanning polytree is a generalised notion of the spanning tree from undirected graphs to directed ones. Each node may have more than one parent nodes. We

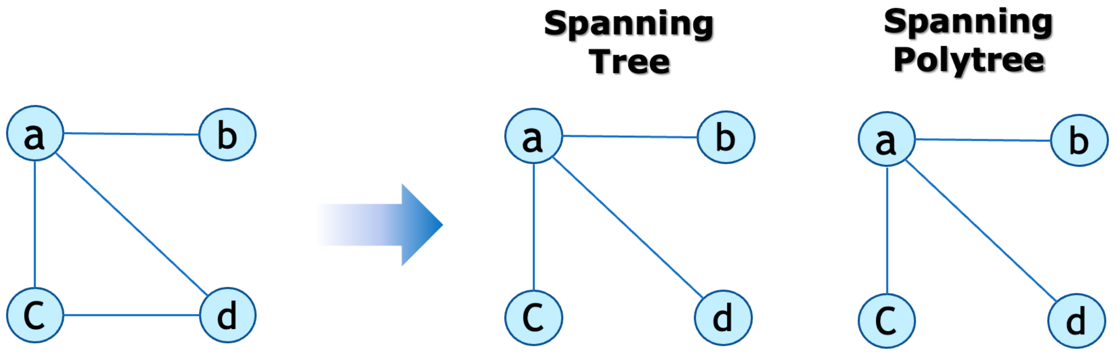


Figure 4.1: Extract a spanning tree and spanning polytree from the undirected G

introduce the spanning polytree because, when G is a directed graph, its traditional spanning tree does not always exist, but a spanning polytree of G always exists.

For instance, in Figure 4.1, the undirected graph G has both a spanning tree and a spanning polytree. One of its spanning trees is precisely the same as the spanning polytree. According to the general properties of a spanning tree (G6), the spanning tree of the undirected graph G is not unique. In Figure 4.1, there is another spanning polytree T' of G (*e.g.*, $< b - a - d - c >$, $< b - a - c - d >$).

However, after the undirected graph is converted to a directed graph, the spanning tree will not always exist. as shown in Figure 4.2.

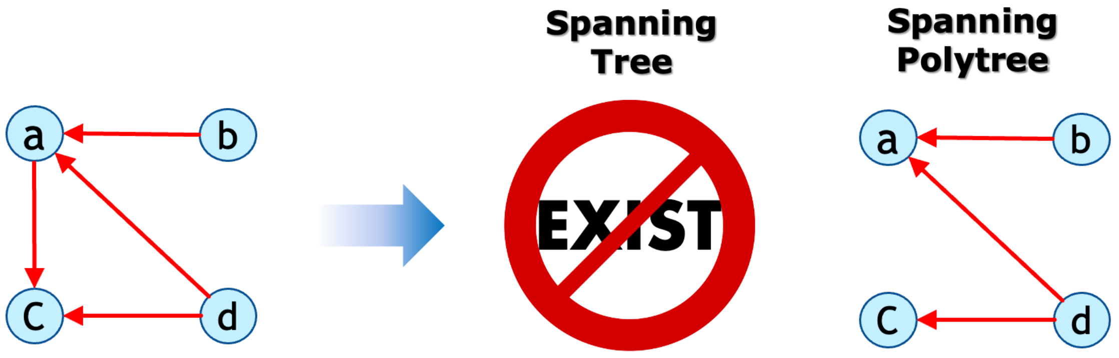


Figure 4.2: Extract a spanning tree and spanning polytree from the directed G

Based on the operation of spanning trees, there is no subgraph of directed graph G that can cover all vertices with a maximum set of edges of G that contains no directed cycles. There are no conventional trees that span G, but one can find a spanning polytree T that spans G. Figure 4.2 clearly shows the difference between spanning tree and spanning polytree, and explains why a spanning polytree is irreplaceable in a similarity search algorithm over large-scale graphs.

If G is an undirected graph, the spanning polytree in Definition 6 reduces to a traditional spanning tree.

To identify a spanning polytree T over a given graph G, we devise a fast heuristic approach based on breadth-first search (BFS) in Procedure 1. Our central idea is as follows. First, we set v to the node with the maximum out-degree in G, start with v as root, push v into an empty queue, and initialise T to empty set. Next, for each iteration, we dequeue v, add the unvisited incoming and outgoing edges of v into T, enqueue the unvisited in- and out-neighbours of v, and then set v to the next node in queue. The iteration continues until the queue becomes empty. Finally, the resulting output T is a spanning polytree of G. The complexity of Procedure 1 is dominated by the BFS search, which is $O(n+m)$ time and $O(n+m)$ memory on a graph with n nodes and m edges.

```
Procedure 1: Find_Spanning_PolyTree ($G$)
   Input  : a graph $G$.
   Output: a spanning polytree $T$ of $G$.
 2 initialise $T := \varnothing$ ;
 4 foreach weakly connected component $G_{wcc} \subseteq G$ do
 6     $U.Enqueue$(a node of maximum out-degree in $G_{\text{wcc}}$);
 8     while $U \neq \varnothing$ do
10         set node $v := U.Dequeue()$ ;
12         $T := T \cup \{$unvisited in- and out-links of $v\}$ ;
14         forall $w \in$ {unvisited in- and out-neighbors of $v$} do  $U.Enqueue(w)$;
16 return $T$ ;
```

4.3 Computing CoSimRank Scores of Spanning Polytrees

Apart from supporting the quick dynamic CoSimRank retrieval on evolving graphs, D-CoSim, presented in the previous chapter, can also be applied to static graphs for accelerating CoSimRank search. Based on D-CoSim, we next propose an efficient scheme, F-CoSim, that dramatically speeds up CoSimRank search over static graphs. Given a static graph G and a query set Q, F-CoSim retrieves the CoSimRank scores $\mathbf{S}[:, Q]$ over G based on three ideas. First, we propose a quick method to find a "spanning polytree" T of G so that G decomposes into $G = T \oplus (G \ominus T)$;

this can be viewed as the old T plus its update $(G \ominus T)$.

Key Observation. Due to its particular "polytree" structure, we notice that the CoSimRank scores are relatively easier to compute. Therefore, we propose a novel and quicker algorithm to retrieve the CoSimRank scores $\mathbf{S}(T)[:, Q]$ over the "spanning polytree". Finally, to compute the CoSimRank scores $\mathbf{S}$ over G, we regard T as the old graph and $G \ominus T$ as the updated graph to T, and then apply our dynamic D-CoSim to compute $\mathbf{S}(T)$ changes in response to the graph update $(G \ominus T)$. With the above, F-CoSim enables a notable speed-up in the CoSimRank search over static graphs, which is achieved by our efficient method to retrieve $\mathbf{S}(T)[:, Q]$ over the "spanning polytree" and our quick D-CoSim to compute the changes to $\mathbf{S}(T)$ with respect to $(G \ominus T)$. We shall elaborate on these ideas in the following pages.

Having identified the spanning polytree T of graph G, we can decompose G into two parts: $G = T \oplus (G \ominus T)$. Due to the particular acyclic structure of T, there is a more efficient way to retrieve the CoSimRank scores of the spanning polytree T. Our key observation is that, if the nodes of T are organised in level order, the adjacency matrix $\mathbf{A}$ of T will exhibit a block superdiagonal structure, which leads to the CoSimRank scores of T, $\mathbf{S}(T)$, displaying a block diagonal structure. Consequently, any two nodes at different levels of T have zero CoSimRank scores. Moreover, the CoSimRank scores of the nodes at the same level of T can be immediately derived from those at the previous level, based on the following theorem:

Theorem 12 (**CoSimRank on Polytree** T)**.** *Given a polytree T with nodes organised in level order, let n_l be the number of nodes at level l $(l = 1, \cdots, L)$. The CoSimRank scores of T, $\mathbf{S}(T)$, are computed level by level as follows:*

$$\begin{gathered}\mathbf{S}(T) = diag(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_L) \quad with \quad \mathbf{S}_1 = \mathbf{I}_{n_1} \quad and \\ \mathbf{S}_l = C\mathbf{A}_{l-1,l}^T \mathbf{S}_{l-1} \mathbf{A}_{l-1,l} + \mathbf{I}_{n_l} \quad (l = 2, \cdots, L)\end{gathered} \tag{4.1}$$

where L is the number of levels in T; $\mathbf{S}(T)$ is a diagonal block matrix with each block $\mathbf{S}_l$ being the CoSimRank scores of n_l^2 pairs of nodes at level l; $\mathbf{A}_{l-1,l}$ is the $(n_{l-1} \times n_l)$ column-normalised adjacency matrix of the subgraph between level $(l-1)$ and level l of T; and $\mathbf{I}_{n_l}$ is the $n_l \times n_l$ identity matrix.

Proof. Since T is a polytree, two surfers starting at different levels cannot meet at a common node through equal-length steps. Thus, only node-pairs at the same level have non-zero scores, leading to the diagonal block structure of $\mathbf{S}(T)$.

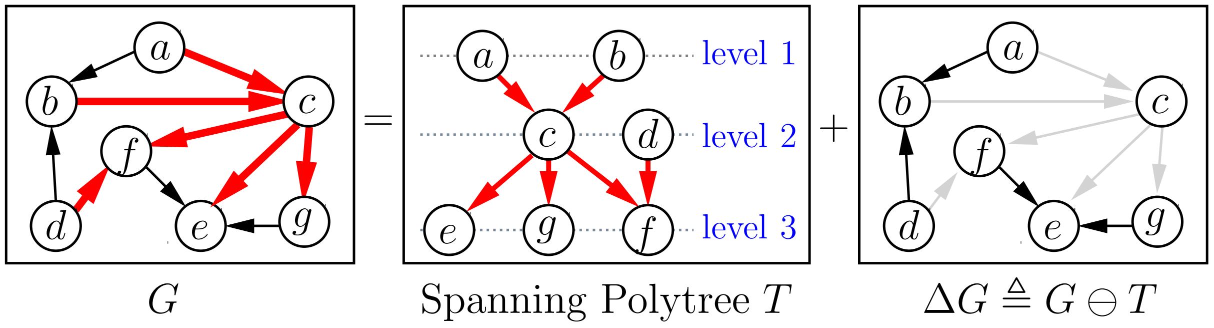


Figure 4.3: Decompose G into a spanning polytree T and ΔG $(= G \ominus T)$

To compute l-th diagonal block $\mathbf{S}_l$, by $\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I}$, we have

$$C \begin{bmatrix} \mathbf{0} & \mathbf{A}_{1,2} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{A}_{L-1,L} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}^T \begin{bmatrix} \mathbf{S}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{S}_L \end{bmatrix} \begin{bmatrix} \mathbf{0} & \mathbf{A}_{1,2} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \mathbf{A}_{L-1,L} \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix} + \begin{bmatrix} \mathbf{I}_{n_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{n_2} & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I}_{n_L} \end{bmatrix}$$

$$= \begin{bmatrix} \mathbf{I}_{n_1} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & C\mathbf{A}_{1,2}^T\mathbf{S}_1\mathbf{A}_{1,2} + \mathbf{I}_{n_2} & \cdots & \mathbf{0} \\ \vdots & \ddots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & C\mathbf{A}_{L-1,L}^T\mathbf{S}_{L-1}\mathbf{A}_{L-1,L} + \mathbf{I}_{n_L} \end{bmatrix} = \underbrace{\begin{bmatrix} \mathbf{S}_1 & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_2 & \cdots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{S}_L \end{bmatrix}}_{diag(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_L)}$$

Since the corresponding diagonal blocks are equal in the last equality, Eq.(4.1) holds. □

Theorem 12 gives a quick and accurate approach for a CoSimRank search on a spanning polytree in a level-by-level style. The CoSimRank scores at level l are immediately computed from those at level $(l-1)$. Retrieving each block $\mathbf{S}_l$ at level l via Eq.(4.1), only requires $O(n_l(n_{l-1} + n_l))$ time and $O(n_l^2)$ memory, as opposed to the original method, which entails $O(K(m+n)n_l)$ time and $O(m+n)$ memory to retrieve n_l^2 pairs of $\mathbf{S}_l$ scores. Since $n_l \ll n = n_1 + n_2 + \cdots + n_L$, the complexity improvement of our approach is significant.

Example 12. *Consider the spanning polytree T in Figure 4.3. Theorem 12 computes the CoSimRank $\mathbf{S}(T)$ of T as follows:*

As Level 1 of T has two nodes $\{a, b\}$, Eq.(4.1) initialises as:

$$\mathbf{S}_1 = \begin{array}{c|cc} & (a) & (b) \\ \hline (a) & 1 & 0 \\ (b) & 0 & 1 \end{array}$$

Since $\mathbf{A}_{1,2} = \left[\begin{smallmatrix} .5 & 0 \\ .5 & 0 \end{smallmatrix}\right]$ and $\mathbf{A}_{2,3} = \left[\begin{smallmatrix} 1 & .5 & 1 \\ 0 & .5 & 0 \end{smallmatrix}\right]$, the CoSimRank similarity of nodes $\{c, d\}$ at Level 2 is computed from $\mathbf{S}_1$:

$$\mathbf{S}_2 = 0.6\mathbf{A}_{1,2}^T\mathbf{S}_1\mathbf{A}_{1,2} + \mathbf{I}_2 = \begin{array}{c|cc} & (c) & (d) \\ \hline (c) & 1.3 & 0 \\ (d) & 0 & 1 \end{array}$$

Next, the CoSimRank similarity of nodes $\{e, f, g\}$ at Level 3 is computed from $\mathbf{S}_2$:

$$\mathbf{S}_3 = 0.6\mathbf{A}_{2,3}^T\mathbf{S}_2\mathbf{A}_{2,3} + \mathbf{I}_3 = \begin{array}{c|ccc} & (e) & (f) & (g) \\ \hline (e) & 1.78 & 0.39 & 0.78 \\ (f) & 0.39 & 1.34 & 0.39 \\ (g) & 0.78 & 0.39 & 1.78 \end{array}$$

Thus, the CoSimRank scores $\mathbf{S}(T) = diag(\mathbf{S}_1, \mathbf{S}_2, \mathbf{S}_3)$. □

4.4 F-CoSim over Large-Scale Network

After CoSimRank $\mathbf{S}(T)$ of the polytree T is computed, F-CoSim next computes the CoSimRank changes $\mathbf{\Delta S}$ with respect to the graph $\Delta G\ (= G \ominus T)$, by utilising our dynamical D-CoSim algorithm defined in Chapter 3. Finally, the two parts ($\mathbf{S}(T)$ and $\mathbf{\Delta S}$) are added together, which produces the CoSimRank $\mathbf{S}$ of the original graph G, *i.e.,* $\mathbf{S} = \mathbf{S}(T) + \mathbf{\Delta S}$.

Algorithm. To optimise the CoSimRank search over static graphs, Theorem 12 is incorporated to D-CoSim. Precisely, to compute the CoSimRank scores $\mathbf{S}$ over G, we first apply Procedure 1 to find a spanning polytree T over G, which decomposes G into $G = T \oplus (G \ominus T)$. Then, by

Theorem 12, we compute CoSimRank scores $\mathbf{S}_T$ over T. Finally, we use our dynamic algorithm D-CoSim to compute the changes $\mathbf{\Delta S}$ to $\mathbf{S}_T$ in response to the graph update $G \ominus T$. The resulting scores ($\mathbf{\Delta S} + \mathbf{S}_T$) are the CoSimRank scores $\mathbf{S}$ over G.

Based on the above, Algorithm 2 provides our complete static scheme, F-CoSim, which incorporates Theorem 12 and our dynamic D-CoSim algorithm. F-CoSim, thus, consists of three phases:

(i) Finding a spanning polytree T over G (line 2),

(ii) Retrieving CoSimRank $\mathbf{S}(T)$ on T (lines 4–15),

(iii) Computing CoSimRank changes $\mathbf{\Delta S}$ in answer to $(G \ominus T)$ (lines 17–21).

Algorithm 2: F-CoSim (G, C, Q, K)

```
Input  : graph $G$, decay factor $C$,
         query set $Q$, #-iteration $K$
Output: CoSimRank scores $\mathbf{S}[:, Q]$ in $G$.
2  set $T :=$ Find_Spanning_PolyTree($G$)
4  set $L_Q :=$ maximum level of the query node of $Q$ in $T$
6  initialise $\mathbf{S}_1 = \mathbf{I}_{n_1}$
8  for $l = 2$ to $L_Q$ do
10     set $n_l :=$ the number of nodes at level $l$ in $T$
12     $\mathbf{A}_{l-1,l}$ is $(n_{l-1} \times n_l)$ col-normalised adjacency block:
13     $\mathbf{A}_{l-1,l}[i,j] = 1/\deg_j^-$ if $\exists(i \to j) \in T$; or 0 otherwise
15     compute $\mathbf{S}_l := C\mathbf{A}_{l-1,l}^T \mathbf{S}_{l-1} \mathbf{A}_{l-1,l} + \mathbf{I}_{n_l}$
17 compute $\mathbf{\Delta S}[:, Q] :=$ D-CoSim $(T, G \ominus T, C, Q, K)$
19 compute $\mathbf{S}[:, Q] = diag(\mathbf{S}_1, \mathbf{S}_2, \cdots, \mathbf{S}_L)[:, Q] + \mathbf{\Delta S}[:, Q]$;
21 return $\mathbf{S}[:, Q]$;
```

Example 13. *Recall G in Figure 4.3. To retrieve $\mathbf{S}[:, c]$ on G, F-CoSim first decomposes $G = T \oplus (G \ominus T)$. Then, it computes the CoSimRank $\mathbf{S}(T)[:, c]$ of T, as shown in Example 12:*

$$\mathbf{S}(T)[:, c] = \begin{array}{c c} & \begin{array}{ccccccc} (a) & (b) & (c) & (d) & (e) & (f) & (g) \end{array} \\ (c) & \left[\begin{array}{ccccccc} 0 & 0 & 1.3 & 0 & 0 & 0 & 0 \end{array} \right]^T \end{array}$$

Next, F-CoSim invokes D-CoSim (Line 17) to compute the CoSimRank increment

$\mathbf{\Delta S}[:, c]$ *with respect to delta graph* $(G \ominus T)$*:*

$$\mathbf{\Delta S}[:, c] = [0, .15, .045, 0, .03, .0225, .045]^T.$$

To save space, we omit the calculation details here. The complete schema of D-CoSim *can be found in Chapter 3.*

Finally, the CoSimRank score $\mathbf{S}[:, c]$ *of* G *(Line 19) is*

$$\mathbf{S}[:, c] = \mathbf{S}(T)[:, c] + \mathbf{\Delta S}[:, c] = [0, .15, 1.345, 0, .03, .0225, .045]^T \, \square$$

Correctness. Next, we show that F-CoSim *correctly* returns the CoSimRank scores $\mathbf{S}[:, Q]$ on G.

Theorem 13. *Given graph* G*, the resulting* $\mathbf{S}$ *returned by Line 21 of* F-CoSim *is the correct CoSimRank scores over* G*.*

Proof. Let $\mathbf{S}(T)$ be the CoSimRank scores of the polytree T, and $\mathbf{A}(T)$ be the column-normalised adjacency matrix of T. According to Line 2 of F-CoSim, after T is retrieved from G, the column-normalised adjacency matrix $\mathbf{A}$ of G is decomposed into two parts ($\mathbf{A}(T)$ and $\mathbf{\Delta A}$):

$$\mathbf{A} = \mathbf{A}(T) + \mathbf{\Delta A} \quad \text{where} \quad \mathbf{\Delta A} \triangleq \mathbf{A} - \mathbf{A}(T) \tag{4.2}$$

Theorem 12 guarantees that the CoSimRank score of T, denoted as $\mathbf{S}(T)$ ($\triangleq diag(\mathbf{S}_1, \cdots, \mathbf{S}_L)$), obtained by Lines 4–15 of F-CoSim, is the correct CoSimRank of T, *i.e.,* $\mathbf{S}(T)$ satisfies the CoSimRank definition:

$$\mathbf{S}(T) = C\mathbf{A}(T)^T\mathbf{S}(T)\mathbf{A}(T) + \mathbf{I} \tag{4.3}$$

where $\mathbf{S}(T) \triangleq diag(\mathbf{S}_1, \cdots, \mathbf{S}_L)$.

Moreover, our correctness proof of D-CoSim in Theorem 12 guarantees that, by viewing T as the old graph, and $(G \ominus T)$ as the graph update to T, the value of $\mathbf{\Delta S}$ from calling D-CoSim (Line 17 of F-CoSim) is the correct CoSimRank increments with respect to the update $(G \ominus T)$ to T. This means that $\mathbf{\Delta S}$ satisfies CoSimRank definition:

$$\begin{aligned}\mathbf{S}(T) + \mathbf{\Delta S} = C \cdot (\mathbf{A}(T) + \mathbf{\Delta A})^T \cdot (\mathbf{S}(T) + \mathbf{\Delta S}) \cdot \\ \cdot (\mathbf{A}(T) + \mathbf{\Delta A}) + \mathbf{I}\end{aligned} \tag{4.4}$$

According to Line 19, the CoSimRank returned by F-CoSim is:

$$\mathbf{S} = diag(\mathbf{S}_1, \cdots, \mathbf{S}_L) + \mathbf{\Delta S} = \mathbf{S}(T) + \mathbf{\Delta S} \tag{4.5}$$

Plugging Eqs.(4.3) and (4.4) into (4.5) produces

$$\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I}.$$

Thus, $\mathbf{S}$ satisfies the CoSimRank definition, *i.e.,* $\mathbf{S}$, returned by F-CoSim, is the correct CoSimRank of G. $\square$

Complexity. Given a query set Q, the time of F-CoSim in each of the three phase is $O(n+m)$, $O(\sum_{l=2}^{L_Q} n_l(n_{l-1}+n_l))$, and $O(K(m+np_{\ominus}|Q|))$, respectively, where n and m are the number of nodes and edges of the graph, respectively, L_Q is the maximum level of the query node Q in T, and $p_{\ominus}$ is the number of update pieces in ΔG. Since $L_Q \leq L \ll n$, $n_{l-1}+n_l \ll n$, and $p_{\ominus} \ll m-n$ in practice, it requires $O(n \max_{2\leq l\leq L_Q}\{n_{l-1}+n_l\} + K(m+np_{\ominus}|Q|))$ time in total, as opposed to the $O(Kn(m+|Q|n))$ time of the original method, to assess $n \times |Q|$ pairs of scores $\mathbf{S}[:,Q]$.

The memory space of F-CoSim in each phase is $O(m+n)$, $O(m+\sum_{l=1}^{L_Q} {n_l}^2)$, and $O(m+Kn)$, respectively. Thus, the total memory is bounded by $O(m+(K+\max_{1\leq l\leq L_Q}\{n_l\})n)$. It can be seen that F-CoSim requires less memory and time compared to the existing CoSimRank algorithm. In the next section, we will present that the performance of F-CoSim can be further optimized.

4.5 Further Efficiency Improvement

When F-CoSim is applied in large-scale graphs, we find two parts of the algorithm that can be improved to speed up the algorithm. Inspired by the parallel computing, we also propose an accelerated algorithm based on F-CoSim algorithm.

The first one is the operational method of extracting spanning polytrees from graphs. In Section 4.2, the breadth-first search(BFS) method is used to extract spanning polytrees from graphs. Later in this section, we propose a more efficient heuristic approach to achieve the same goal with less time consumption.

The other part is the second step of the F-CoSim algorithm, i.e., computing CoSimRank scores on polytree T. The final result of the F-CoSim algorithm is a single column with respect to query. Thus, if the result of the CoSimRank score on polytree T is a single source, it will improve the efficiency of F-CoSim. The following subsection will elaborate on these ideas.

4.5.1 The Optimisation of Spanning Polytree Extraction.

To extract spanning polytrees from a graph G, the breadth-first search method or the depth-first search operation methods can be used. The breadth-first search method(BFS) has been used in Procedure 1.

The BFS method is an algorithm for searching a tree structure from a network. The searching starts from a tree root node, and then explores all the nodes at the present depth. This operation is repeated until all the nodes have been explored. The depth-first search (DFS) is the same as the BFS. It starts at the tree root node and traverses a tree or graph from the parent node down to its children and grandchildren nodes in a single path until it reaches a dead end. When there are no more nodes to visit in a path, the DFS algorithm backtracks to a point where it can choose another path to take. It will repeat the process over and over until all nodes have been explored. Both DFS and BFS methods can implement tree extraction well; however, there is another method – Heapsort, which can extract a tree from a graph more efficiently than the two. Since the F-CoSim algorithm needs to calculate the CoSimRank scores of spanning polytrees level by level, it is better to reduce the levels of polytree, which will significantly reduce the repetition of computations, and consequently reduce time consumption.

To identify a spanning polytree over a given graph with fewer polytree levels, we propose a more efficient heuristic approach based on Procedure 1. Here, we adapt the key idea of a heapsort to improve the efficiency of Procedure 1. Heapsort is an advanced selection, comparison-based sorting algorithm. It separates the pool of whole nodes into two parts, sorted and unsorted nodes. The basic idea of a heapsort is to choose the largest node from the unsorted part and insert it into the sorted part. It avoids exploring nodes in the sorted part, which significantly reduces the sort time consumption. When implementing this critical idea of a heapsort in the extraction of a spanning polytree, we choose the node with the most out-neighbours as the polytree root node. We always give priority to the node which has a larger out-degree. These operations repeat until all nodes have been explored.

To identify a spanning polytree T over a given graph G more efficiently, we devise a better heuristic approach based on the heapsort method in Procedure 2. The central idea of Procedure 2 is as follows. First, same as Procedure 1, we set v to the node with the maximum out-degree in G, set the node v as root, push v into a sorted queue, and initialise T set to empty. Next, for each iteration, we dequeue v, add the unsorted incoming and outgoing edges of v into T,

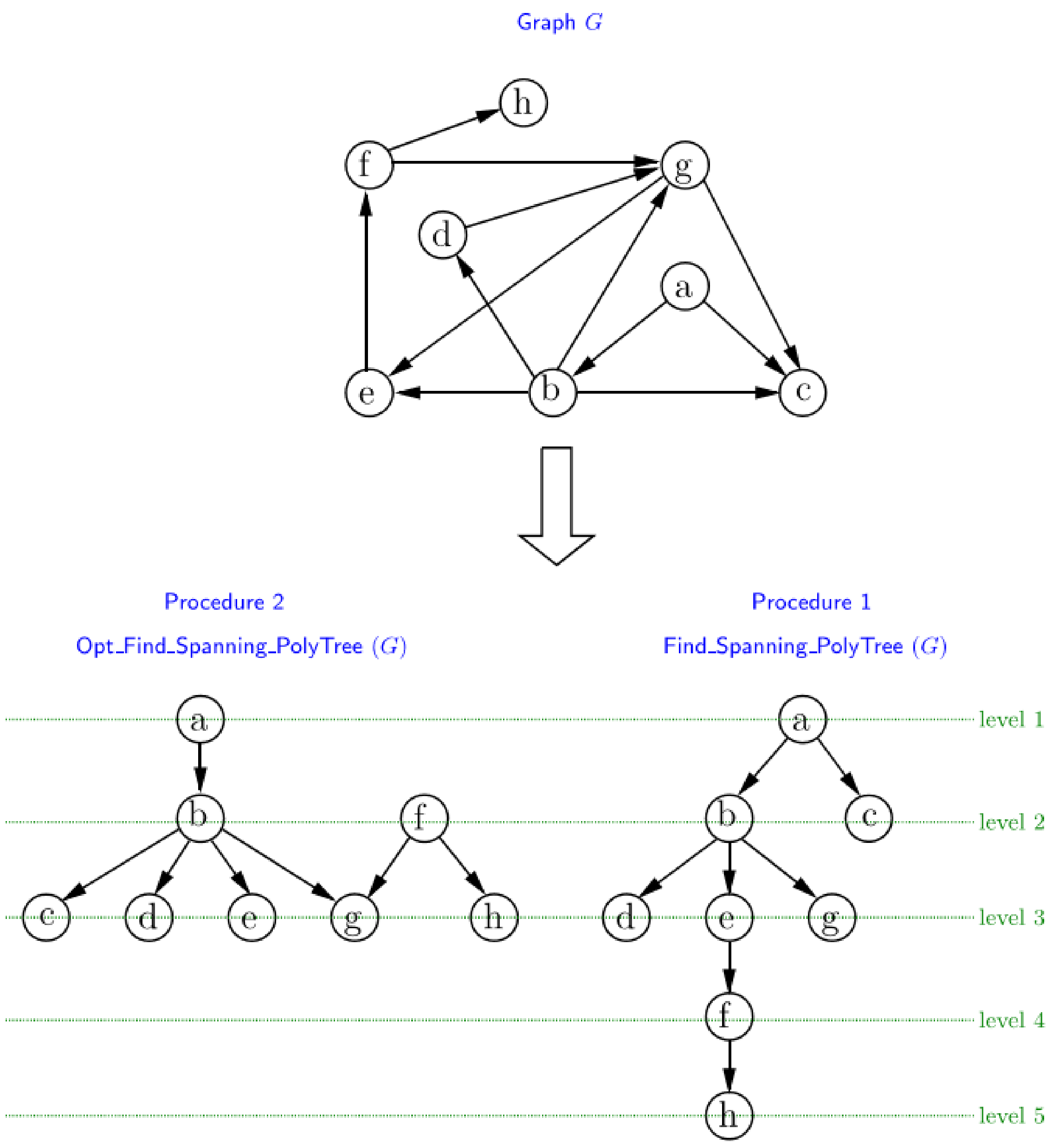


Figure 4.4: Comparison of Opt_Find_Spanning_PolyTree and Find_Spanning_PolyTree Algorithm

enqueue the unvisited in- and out-neighbours of v with the number of in- and out-degree orders, and set v to the next node in queue. The iteration continues until the set U becomes empty. Finally, the resulting output T is a spanning polytree of G. The complexity of Procedure 1 is dominated by the heapsort, which takes $O(nlog_n)$ time on a graph with n nodes and m edges.

Procedure 2: Opt_Find_Spanning_PolyTree (G)

```
Input  : a graph $G$.
Output: a spanning polytree $T$ of $G$.
2   initialise $T := \varnothing$ ;
4   foreach weakly connected component $G_{wcc} \subseteq G$ do
6       $U.Enqueue$(a node of maximum out-degree in $G_{\text{wcc}}$);
8       while $U \neq \varnothing$ do
10          set node $v := U.Dequeue()$ ;
12          $T := T \cup \{$unsorted in- and out-links of $v\}$ ;
14          forall $t$ do h;
15          e $w \in \{$sort unvisited in- and out-neighbors of $v$ with larger in- and out-degree$\}$
             $U.Enqueue(w)$
17  return $T$ ;
```

Take the example from Figure 4.4 to describe the merit of the Opt_Find_Spanning_PolyTree algorithm. In Figure 4.4, there is a given graph G with eight nodes. Extracting spanning polytree T from graph G can be done by the two operations we discussed earlier. The results of the two algorithms are different. The spanning polytree extracted by Procedure 2 has three levels, whereas the spanning polytree extracted using the Find_Spanning_PolyTree algorithm has five levels. According to Theorem 12, the number of polytree levels decides the call number of computing the CoSimRank scores of a spanning polytree algorithm. Therefore, the fewer levels that the polytree has, the fewer CoSimRank scores of a spanning polytree need to be computed, and the lesser computation time will be consumed. At the same time, the number of cut edges of a graph G grows with the increasing number of polytree levels, so we need to call D-CoSim algorithm once for each update brunch.

In conclusion, the Opt_Find_Spanning_PolyTree algorithm, with fewer polytree levels, is more efficient than the Find_Spanning_PolyTree algorithm. For the F-CoSim algorithm, extracting a spanning polytree with fewer levels can reduce the call number of computation algorithms, which could significantly improve its efficiency.

4.5.2 Efficient Single-Source CoSimRank Scores' Retrieval of Spanning Polytree.

After identifying a spanning polytree T of a graph G with the Opt_Find_Spanning_PolyTree algorithm, we can decompose graph G into two parts: $G = T \oplus (G \ominus T)$. According to the design of the F-CoSim algorithm, computing the CoSimRank scores of a spanning polytree requires calculating the similarity scores of all node pairs at each level. However, F-CoSim is an algorithm that evaluates the similarity scores of single-source queries; therefore, some information from all node pairs of the spanning polytree is useless, which means it will reduce the efficiency of the algorithm. To bridge this gap, we propose a novel algorithm to calculate single-source CoSimRank scores of each level of a spanning polytree.

Theorem 14. *Given a graph G, q is a query, and L_q is the level of the spanning polytree where query q located $L_q \leq L$, the single-source CoSimRank score $[S_{L_q}]_{*,q}$ is computed by:*

$$[\mathbf{S}_{L_q}]_{*,q} = \mathbf{v}_{L_q-1} \tag{4.6}$$

where vector $\mathbf{v}$ is iterated as:

$$\begin{cases} \mathbf{v}_0 = \mathbf{u}_{L_q-1} \\ \mathbf{v}_l = C\mathbf{A}_{l,l+1}^T \mathbf{v}_{l-1} + \mathbf{u}_{L_q-l-1} \end{cases} \quad (\forall\ l = 1, \cdots, L_q - 1)$$

and $\mathbf{u}_l$ can be generated by:

$$\begin{cases} \mathbf{u}_0 = \mathbf{e}_q \\ \mathbf{u}_l = \mathbf{A}_{L_q-l,L_q-l+1} \mathbf{u}_{l-1} \end{cases} \quad (\forall\ l = 1, \cdots, L_q - 1)$$

Proof. We want to show that $[\mathbf{S}_{\mathbf{L}_q}]_{*,q}$ obtained from Equation 4.6 is exactly equal to the result of Equation 4.1.

First, Equation 4.6 can be expressed as:

$$[\mathbf{S}_{L_q}]_{*,q} = \mathbf{e}_q + \sum\nolimits_{l=1}^{L_q-1} (C^l \prod\nolimits_{i=1}^{l} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{u}_l)$$

where $\mathbf{u}_l$ is replaced by $\mathbf{A}_{L_q-l,L_q-l+1} \cdots \mathbf{A}_{L_q-1,L_q} \cdot \mathbf{e}_q$.

Next, according to the equation of $\mathbf{v}_l$ from Theorem 14, we continue the iterative calculation $\mathbf{v}_l$, with respect to $\forall\ l = 1, \cdots, L_q - 1$, so it can be written as:

$$\mathbf{v}_{L_q} = \ \mathbf{e}_q + \sum\nolimits_{l=1}^{L_q-1} (C^l \prod\nolimits_{i=1}^{l} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{u}_l)$$

Then, taking the equation $\mathbf{v}_l = C\mathbf{A}_{l,l+1}^T \mathbf{v}_{l-1} + \mathbf{u}_{L_q-l-1}$ with fixed variable l, and multiplying $C^{L_q-l-1} \prod_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T$ on both sides of $\mathbf{v}_l$, it can be shown as:

$$\begin{aligned} C^{L_q-l-1} \prod\nolimits_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_l &= C^{L_q-l-1} \prod\nolimits_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_{l-1} \\ &\quad + \mathbf{c}^{L_q-l-1} \prod\nolimits_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{u}_{L_q-l-1} \end{aligned}$$

Then, $\sum_{l=0}^{L_q-1} (\bullet)$ can be added on both sides, and expressed as:

$$\begin{aligned} &\underbrace{\sum\nolimits_{l=1}^{L_q-1} C^{L_q-l-1} \prod\nolimits_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_l}_{=\mathbf{v}_{L_q+1} + \sum_{l=1}^{L_q-2} C^{L_q-l-1} \prod_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_l} \\ &= \underbrace{\sum\nolimits_{l=1}^{L_q-1} C^{L_q-l} \prod\nolimits_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_{l-1}}_{=\sum_{l=0}^{L_q-2} C^{L_q-l-1} \prod_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_l} + \underbrace{\sum\nolimits_{l=1}^{L_q-1} C^{L_q-l-1} \prod\nolimits_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{u}_{L_q-l-1}}_{=\sum_{l=1}^{L_q-2} C^l \prod_{i=1}^{l} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{u}_l + \mathbf{e}_q} \end{aligned}$$

Finally, we remove $\sum_{l=1}^{L_q-2} C^{L_q-l-1} \prod_{i=1}^{L_q-l-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_l$ from both sides and obtain:

$$\begin{aligned} \mathbf{v}_{L_q-1} &= C^{L_q-1} \prod\nolimits_{i=1}^{L_q-1} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{v}_0 + \sum\nolimits_{l=1}^{L_q-2} C^l \prod\nolimits_{i=1}^{l} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{u}_l + \mathbf{e}_q \\ &= \mathbf{e}_q + \sum\nolimits_{l=1}^{L_q-1} C^l \prod\nolimits_{i=1}^{l} \mathbf{A}_{L_q-i,L_q-i+1}^T \mathbf{u}_l \end{aligned}$$

It is easy to find that this equation is exactly same as

$$\begin{aligned} [\mathbf{S}_{L_q}]_{*,q} &= C\mathbf{A}_{L_q-1,L_q}^T [C\mathbf{A}_{L_q-2,L_q-1}^T [C\mathbf{A}_{L_q-3,L_q-2}^T [\cdots [C\mathbf{A}_{1,2}^T \mathbf{A}_{1,2} \cdots \mathbf{A}_{L_q-1,L_q} \mathbf{e}_q + \\ &\quad \mathbf{A}_{2,3} \cdots \mathbf{A}_{L_q-3,L_q-2} \mathbf{A}_{L_q-2,L_q-1} \mathbf{A}_{L_q-1,L_q} \mathbf{e}_q] + \ \cdots \mathbf{A}_{L_q-3,L_q-2} \mathbf{A}_{L_q-2,L_q-1} \mathbf{A}_{L_q-1,L_q} B_q] \\ &\quad + \mathbf{A}_{L_q-2,L_q-1} \mathbf{A}_{L_q-1,L_q} \mathbf{e}_q] + \mathbf{A}_{L_q-1,L_q} \mathbf{e}_q] + \mathbf{e}_q \\ &= \mathbf{e}_q + \sum\nolimits_{l=1}^{L_q-1} (C^l \prod\nolimits_{i=1}^{l} \mathbf{A}_{L_q-i,L_q-i+1}^T \prod\nolimits_{j=L_q-l}^{L_q-1} \mathbf{A}_{j,j+1} \mathbf{e}_q) \end{aligned}$$

□

Example 14. *Recall Example 12. Here, we take column 2 of the second level of the spanning tree in Figure 4.3 as the query. According to Theorem 14, the CoSimRank scores* $[\mathbf{S_3}]_{*,2}$ *of* T *can be calculated as follows:*

The adjacency matrix of the spanning polytree can be written as: $\mathbf{A}_{1,2} = \left[\begin{smallmatrix} 0.5 & 0 \\ 0.5 & 0 \end{smallmatrix}\right]$ *and* $\mathbf{A}_{2,3} = \left[\begin{smallmatrix} 1 & 0.5 & 1 \\ 0 & 0.5 & 0 \end{smallmatrix}\right]$.

Based on the adjacency matrix and Theorem 14, we apply successive substitution to $\mathbf{v}_l$ *and* $\mathbf{u}_l$*, and yield:*

k	$\mathbf{u}_l$	$\mathbf{v}_l$
0	$\mathbf{u}_0 := \mathbf{e}_q \quad \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$	$\mathbf{v}_0 := \mathbf{u}_2 \quad \begin{bmatrix} 0.25 & 0.25 \end{bmatrix}^T$
1	$\mathbf{u}_1 := \mathbf{A}_{2,3}\mathbf{u}_0 \quad \begin{bmatrix} 0.5 & 0.5 \end{bmatrix}^T$	$\mathbf{v}_1 := C\mathbf{A}_{1,2}\mathbf{v}_0 + \mathbf{u}_1 \quad \begin{bmatrix} 0.65 & 0.5 \end{bmatrix}^T$
2	$\mathbf{u}_2 := \mathbf{A}_{1,2}\mathbf{u}_1 \quad \begin{bmatrix} 0.25 & 0.25 \end{bmatrix}^T$	$\mathbf{v}_2 := C\mathbf{A}_{2,3}\mathbf{v}_1 + \mathbf{u}_0 \quad \begin{bmatrix} 0.39 & 1.34 & 0.39 \end{bmatrix}^T$

where $[\mathbf{S_3}]_{*,2} = \mathbf{v}_2$*. Thus, CoSimRank scores* $[\mathbf{S}_3]_{*,2} = \left[\, 0.39 \; 1.34 \; 0.39 \,\right]^T$*.*

To combine these improvements (i.e., Opt_Find_Spanning_PolyTree algorithm and single-source similarity search of a spanning polytree) with the F-CoSim algorithm, we can obtain a more optimized F-CoSim algorithm. Here, we named it as Opt_F-CoSim.

The optimized algorithm is explained in detail as Procedure 3.

The critical improvements of Opt_Find_Spanning_PolyTree algorithm are:

(i) The Opt_Find_Spanning_PolyTree algorithm takes the place of Find_Spanning_PolyTree algorithm (Line 2). It can extract a spanning polytree T from a graph G with less levels more efficiently. The number of polytree levels decides the number of similarity search algorithm calls over a spanning polytree, which can greatly reduce the computation complicity and improve the efficiency of F-CoSim algorithm.

(ii) We can convert an all pairs similarity search into a single-source similarity search over a spanning polytree. According to the F-CoSim algorithm, after a spanning polytree is extracted from graph G, it calculates all node pairs CoSimRank scores of each level of spanning polytree T. However, to compute the single-source similarity scores of graph G, with respect to query q, the algorithm does not need all node pairs information of the spanning polytree. Thus, calculating the single-source similarity scores of a spanning polytree can greatly reduce redundancy in computation. At the same time, the time complexity of the F-CoSim algorithm can be drastically reduced.

4.5.3 Parallel Computing

Parallel computing is an effective technique to solve extremely large and complex problems with less time consumption (Almasi & Gottlieb, 1994). The critical and fundamental method of parallel computing involves breaking down large questions into smaller and independent parts

```
Algorithm 3: Opt_F-CoSim $(G, C, Q, K)$
  Input  : graph $G$, decay factor $C$,
           query set $Q$, #-iteration $K$
  Output: CoSimRank scores $\mathbf{S}[:, Q]$ in $G$.
2  set $T :=$ Opt_Find_Spanning_PolyTree($G$)
4  set $L_Q :=$ maximum level of the query node of $Q$ in $T$
6  initialise $\mathbf{S}_1 = \mathbf{I}_{n_1}$
8  for $l = 2$ to $L_Q$ do
10     set $n_l :=$ the number of nodes at level $l$ in $T$
12     $\mathbf{A}_{l-1,l}$ is $(n_{l-1} \times n_l)$ col-normalised adjacency block:
13     $\mathbf{A}_{l-1,l}[i, j] = 1/\deg_j^-$ if $\exists(i \to j) \in T$; or 0 otherwise
15 initialise $\mathbf{u}_0 := e_j$
17 for $l = 2$ to $L_Q$ do
18     $\mathbf{u}_l = \mathbf{A}_{L_q-l,L_q-l+1}\mathbf{u}_{l-1}$
20 initialise $\mathbf{v}_0 := \mathbf{u}_{L_Q}$
22 for $l = 2$ to $L_Q$ do
23     $\mathbf{v}_l = C\mathbf{A}_{l,l+1}^T\mathbf{v}_{l-1} + \mathbf{u}_{L_q-l-1}$ free $\mathbf{v}_{l-1}$ and $\mathbf{u}_{L_Q-l-1}$
25 Then $\mathbf{S}(\mathbf{T})[:, Q] = \mathbf{v}_{L_Q}$
27 compute $\mathbf{\Delta S}[:, Q] :=$ D-CoSim $(T, G \ominus T, C, Q, K)$
29 compute $\mathbf{S}[:, Q] = \mathbf{S}(\mathbf{T})[:, Q] + \mathbf{\Delta S}[:, Q]$;
31 return $\mathbf{S}[:, Q]$;
```

that generally have the same properties and computational requirements. Each smaller part of the large problem is simultaneously processed by an independent processor. These processors communicate via shared memory. Finally, the results from each processor are combined as part of the result of the overall problem. Based on the fundamental design of parallel computing, we can find that the most important goal of the technique is to use more independent processors (computation power) to quickly and accurately solve the overall large and complex problem.

Inspired by the definition of parallel computing, we propose a novel algorithm to speed up the F-CoSim algorithm when it is applied over large-scale networks. We name it as F-CoSim-Para algorithm.

The first step of the F-CoSim-Para algorithm is to reasonably decompose a large and complex network into small chunks. In order to do so efficiently, we refer to METIS (Karypis & Kumar, 1998). METIS is a software package, which includes various multilevel algorithms, for efficient and reasonable graph partitioning. The steps of METIS' multilevel algorithms are as follows.

(i) Generate a sequence of graphs to coarsen the large graph. They can be denoted as $(G_0, \cdots, G_i, G_j, \cdots, G_N)$, where $0 \leq i \leq j \leq N$, and G_0 is the original large graph. The number of nodes in G_j is smaller than the number of nodes in G_i.

(ii) Generate a partition of graph G_N.

(iii) Following the order of $(G_N, \cdots, G_1, G_0)$, project the partition back and refine it with each graph respectively.

The final partition, which is obtained from Step (iii), is the graph partition. The implementation method of METIS can be found in Appendix A.2.

Leveraging the graph partition from METIS, the large graph is divided into several independent parts and some cut edges. For the F-CoSim-Para algorithm, the next step is to take the cut edges as graph updates, and each chunk of the original graph as a small graph to compute CoSimRank scores respectively. The CoSimRank scores of each chunk are computed simultaneously (parallel computing). This step greatly increases the time efficiency of the F-CoSim algorithm, since it adds computational power to execute the CoSimRank score of each chunk simultaneously. Then all CoSimRank score matrices are put into the diagonal matrix, which is the CoSimRank scores of the original graph without cut edges.

The last step of the F-CoSim-Para algorithm is to use the D-CoSim algorithm to calculate the updated CoSimRank scores with respect to the cut edges.

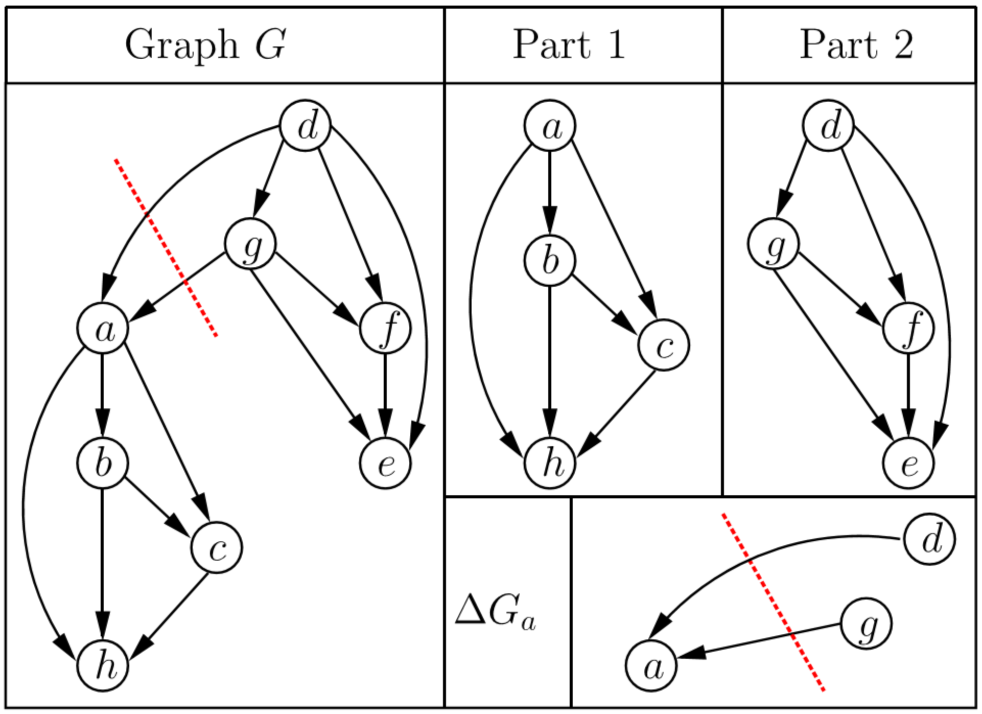


Figure 4.5: Parallel Computing

Example 15. *In Figure 4.5, given is a graph G with eight nodes; take node e as query. Decay factor $C = 0.6$ and iteration number $K = 4$. Following* F-CoSim*-Para algorithm, the CoSimRank scores of the graph can be retrieved as follows.*

First, follow the METIS method to separate the graph into two partitions (i.e., part1 and part2); and the cut edges are $\langle (d, g) \to a \rangle$. It can be seen from Figure 4.5 that the two partitions of the graph are: part1 $= [\, a\ b\ c\ h \,]'$, and part2 $= [\, d\ e\ f\ g \,]'$. Then, based on the graph's partition, the adjacency matrix of each part can be expressed as:

$$\mathbf{A}_{part1} = \begin{bmatrix} 0 & ^1/_3 & ^1/_2 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & ^1/_3 & 0 & 0 \\ 0 & ^1/_3 & ^1/_2 & 0 \end{bmatrix}, \quad \mathbf{A}_{part2} = \begin{bmatrix} 0 & 1 & ^1/_2 & ^1/_3 \\ 0 & 0 & ^1/_2 & ^1/_3 \\ 0 & 0 & 0 & ^1/_3 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

Then, distributing the computation of CoSimRank scores of each part to different and independent processors and performing the tasks in parallel, we obtain:

$$\mathbf{S}_{part1} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.306 & 0.29 & 0.2 \\ 0 & 0.29 & 1.39 & 0.3 \\ 0 & 0.2 & 0.3 & 1.6 \end{bmatrix} \quad \mathbf{S}_{part2} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1.6 & 0.3 & 0.2 \\ 0 & 0.3 & 1.39 & 0.29 \\ 0 & 0.2 & 0.29 & 1.306 \end{bmatrix}$$

Leveraging the CoSimRank scores of each part, the two CoSimRank score matrices are put into a diagonal matrix, which is the similarity search result of the original graph without the cut edges.

It is

$$\mathbf{S}_{part} = \begin{bmatrix} \mathbf{S}_{part1} & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_{part2} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1.306 & 0.29 & 0.2 & 0 & 0 & 0 & 0 \\ 0 & 0.29 & 1.39 & 0.3 & 0 & 0 & 0 & 0 \\ 0 & 0.2 & 0.3 & 1.6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1.6 & 0.3 & 0.2 \\ 0 & 0 & 0 & 0 & 0 & 0.3 & 1.39 & 0.29 \\ 0 & 0 & 0 & 0 & 0 & 0.2 & 0.29 & 1.306 \end{bmatrix}$$

Furthermore, follow the D-CoSim algorithm to calculate $\Delta\mathbf{S}$ with respect to the cut edges' chunk $\Delta G_a = \langle (d,g) \to a \rangle$ in answer to query e,

$$\Delta\mathbf{S}[:,e] = \sum_{k=0}^{3} C^k (\mathbf{p}^{(k)}[e](\mathbf{t}^{(k)})^T + \mathbf{t}^{(k)}[e](\mathbf{p}^{(k)})^T) = \begin{bmatrix} 0 & 0.29 & 0.39 & 0.3 & 0.39 & 0.09 & 0.045 & 0.03 \end{bmatrix}^T$$

Finally, the CoSimRank scores of the original graph can be calculated by adding $\mathbf{S}_{part}$ and $\Delta\mathbf{S}$,

$$\mathbf{S}[:,e] = \mathbf{S}_{part}[:,e] + \Delta\mathbf{S}[:,e] = \begin{bmatrix} 0 & 0.29 & 0.39 & 0.3 & 1.39 & 0.09 & 0.045 & 0.03 \end{bmatrix}^T$$

4.6 Experimental Evaluation

In this section, we present an experimental study on actual and large-scale datasets to evaluate the advantages of our similarity search algorithm, F-CoSim. Furthermore, we evaluate the Opt_F-CoSim algorithm (F-CoSim with the two optimisations) on realistic and large-scale datasets. Three metrics evaluate the performance efficiency:

(a) Running Time. On static graphs, F-CoSim shows more efficiency than the best-known CoSimRank approach. Opt_F-CoSim takes less time than F-CoSim on networks.

(b) Memory Space. F-CoSim and Opt_F-CoSim only require linear memory and scale well on million-node graphs.

(c) Accuracy. F-CoSim and Opt_F-CoSim do not compromise any accuracy for the speedup.

4.6.1 Experimental Setting

Datasets. We implement our similarity search algorithms over three real datasets, including: ca-HepPh (HP), email-EuAll (EE) and wiki-Talk (WT). Apart from these three datasets, we also evaluate Procedure 2 on as-735 (AS), web-Google (WG) and soc-LiveJournal (LJ). Thus, there are six realistic datasets involved in this chapter. The size of each dataset has been illustrated in Table 3.2. The first dataset, AS, is an undirected graph, whereas the others are directed graphs. Table 3.2 shows that the size of datasets is sorted from small to large. The smallest graph is AS with 7716 nodes and 26467 edges, and the most extensive graph is LJ, which has more than a million nodes and edges. More details of each dataset can be found in Section 3.4.1.

All experiments are conducted on a PC with an Intel Core i7-6700 3.40GHz CPU and 64GB memory compiled by VC++.

Compared Algorithms. We implement F-CoSim and Opt_F-CoSim over static graphs, and compare it with two state-of-art CoSimRank competitors:

(a) CSR, a method by (Rothe & Schütze, 2014) that retrieves a CoSimRank score from the sum of the dot product of two Personalised PageRank vectors; and

(b) CSM, a repeated-squaring method by (Yu & McCann, 2015a) that cuts down the number of CoSimRank iterations.

Parameters. We chose the following parameters by default:

(a) the decay factor $C = 0.8$,

(b) the number of iterations $K = 5$,

as previously used in (Rothe & Schütze, 2014).

4.6.2 Experimental Results

In this section, we present the results of the experiments to show the superiority of our algorithm, F-CoSim over realistic and large-scale graphs. Then we show the experimental results of the optimisation algorithms F-CoSim. The performance efficiency is separated into three aspects: time efficiency, memory efficiency and accuracy.

4.6.2.1 Experimental Results of F-CoSim

In this subsection, we compare the F-CoSim algorithm with state-of-the-art similarity search algorithms over three large-scale datasets. The results have been separated into three parts: time efficiency, memory efficiency and accuracy.

Time Efficiency. We first implement our algorithm F-CoSim and baseline algorithms on three realistic datasets to compare their time efficiency.

Figure 4.6 shows the time efficiency of F-CoSim on static graphs. We report that the results on three datasets, and the trends on other datasets are similar. Figure 4.6 compares the time of F-CoSim with CSR and CSM on each dataset. We discern that F-CoSim consistently outperforms CSR by speedup of up to 9.8x (on EE). Thus, using our spanning polytree for a quick CoSimRank search is effective (Theorem 13). Moreover, CSM is the fastest algorithm on HP dataset, but this method only survives on small-scale graphs due to its high memory storage requirement on

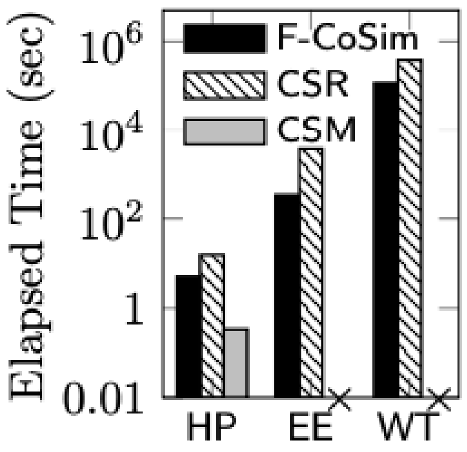


Figure 4.6: Time Efficiency on Static Graph

account of repeated squaring. In contrast, F-CoSim scales well on million-edge graphs (*e.g.,* WT EE).

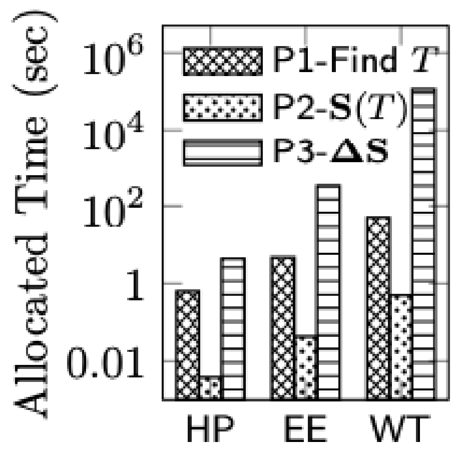


Figure 4.7: Time Efficiency: Phases in F-CoSim

Since F-CoSim encompasses three phases (Algorithm 2), Figure 4.7 details the time allocated in each phase per dataset. We see that among these phases, Phase 2 (computing $\mathbf{S}(T)$ on spanning polytree T) takes the smallest portion; Phase 1 (finding T from G), the second smallest; and Phase 3 (computing $\mathbf{\Delta S}$ with respect to $G \ominus T$), the largest. This agrees well with our complexity analysis of Algorithm 2, where the time of Phase 2, $O(\sum_{l=2}^{L_Q} n_l(n_{l-1} + n_l))$, is independent of the graph size n, unlike Phases 1 and 3 that hinge on n ($\gg n_l$).

Memory Efficiency. This subsection shows the memory efficiency of F-CoSim and the other baseline algorithms, and the memory cost of each step of the F-CoSim algorithm.

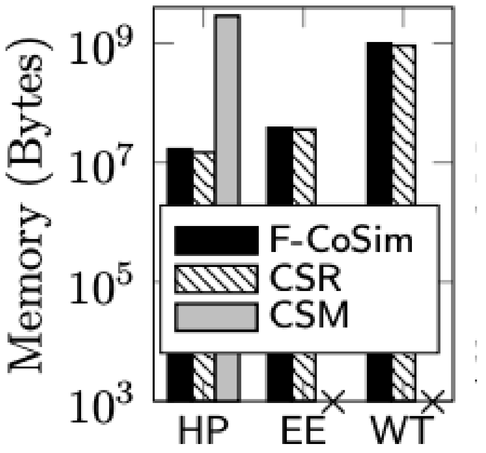


Figure 4.8: Memory Efficiency & Scalability of F-CoSim

Figure 4.8 depicts the memory efficiency of F-CoSim on three real datasets in comparison with CSR and CSM. On each dataset, we randomly select $|Q| = 500$ queries. Figure 4.8 shows the average memory cost in response to one query. The memory of F-CoSim on static graphs shows a similar tendency. F-CoSim and CSR show comparable memory efficiency on the three datasets, however, CSM costs 2 order-of-magnitude memory space than F-CoSim on HP. CSM cannot be implemented on EE and WTbecause of the expensive memory cost. In contrast, F-CoSim works well on the large-scale datasets (EE and WT), which illustrates the scalability of F-CoSim.

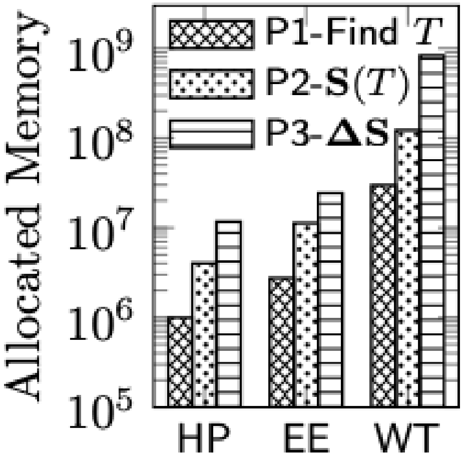


Figure 4.9: Memory Efficiency: Phases in F-CoSim

Figure 4.9 shows the memory usage at each phase of F-CoSim on each dataset. We use abbreviations to denote the three steps of the F-CoSim algorithm separately. P1-Find T is the first step of the F-CoSim algorithm, which is extracting spanning spanning polytree from graphs. P2-$\mathbf{S}(T)$ is the second step, which involves computing the CoSimRank scores of a spanning polytree. P3-$\Delta\mathbf{S}$ is the last step, which is the computation of $\Delta\mathbf{S}$. From the results, we see that Phase 1 (finding T) has the lowest memory as it is based on a linear BFS search. Phase 2 (computing $\mathbf{S}(T)$ on T) requires a larger memory than Phase 1 to store the resulting $\mathbf{S}(T)[:, Q]$ due to its overheads. Phase 3 takes the most memory space to generate $\Delta\mathbf{S}$ using the D-CoSim algorithm. These findings agree with our memory analysis in Algorithm 2.

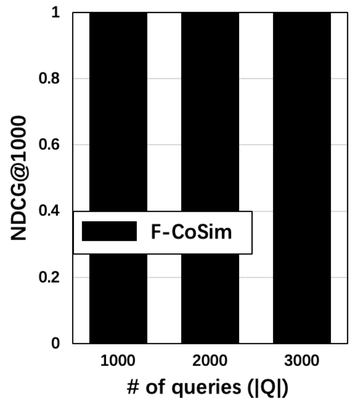


Figure 4.10: Memory Efficiency: Phases in F-CoSim

Accuracy. We evaluate the accuracy of F-CoSim, relative to the original CSR, on real datasets. We randomly pick various query sets with its size $|Q|$ varying from 1000 to 3000. For each query set Q, based on the CoSimRank scores $\mathbf{S}[:, Q]$ from F-CoSim, we measure the results of their similarity ranking via NDCG (Normalised Discounted Cumulative Gain) (Y. Wang et al., 2013):

$$\text{NDCG}_Q@k = \tfrac{1}{|Q|} \sum_{q=1}^{|Q|} \left(Z_{k,j} \sum_{x=1}^{k} \tfrac{2^{\mathbf{S}[x,q]}-1}{\log_2(1+x)} \right)$$

where $Z_{k,j}$ is a normalisation factor that is the DCG ranking result obtained using the original method of CSR. Thus, NDCG = 1 implies that the CoSimRank ranking of the compared algorithm perfectly matches that of CSR, with no accuracy loss. Figure 4.10 shows the accuracy of F-CoSim via NDCG for top $k = 1000$ CoSimRank ranking scores on AS. The trends on other datasets are similar to this. To save space, we omit these trends here. From the results, we notice that for each query set Q, the NDCG of F-CoSim is 1, thus implying that F-CoSim does not sacrifice any accuracy for its speedup. This verifies the correctness of Theorem 12.

4.6.2.2 Experimental Results of the Optimisation F-CoSim Algorithm

In this subsection, we evaluate the optimisation F-CoSim algorithm, Opt_F-CoSim, and demonstrate that it can greatly outperform F-CoSim on large-scale datasets on account of the two optimization technologies we adopted. The performance efficiency is also evaluated from three aspects: time efficiency, memory efficiency and accuracy.

Time Efficiency. We evaluate Procedure 2, single-source similarity search on a spanning polytree algorithm and Algorithm 3 on realistic datasets.

We first implement Procedure 1 and Procedure 2 on six realistic and large datasets to compare their time efficiency.

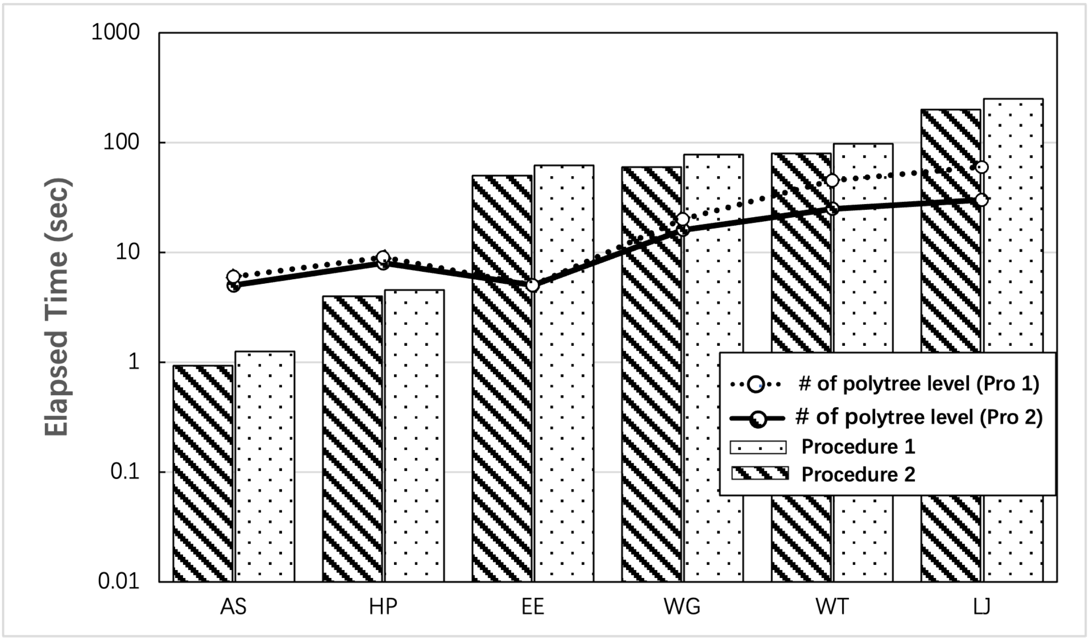


Figure 4.11: Time Efficiency Comparison of Procedure 1 and Procedure 2

Figure 4.11 shows the time cost of Procedure 1 (Find_Spanning_Polytree(G)) and Procedure 2 (Opt_Find_Spanning_Polytree(G)) over six datasets. The bar chart in Figure 4.11 shows the time cost of the two procedures, and the line chart shows the number of polytree levels of two procedures. The Opt_Find_Spanning_Polytree(G) procedure is 1.3x faster than the Find_Spanning_Polytree(G) procedure on AS, and the remaining datasets show a similar trend. It was also found that Procedure 2 is quicker than Procedure 1 in terms of performance on all six datasets. Moreover, the number of polytree levels affects the time efficiency of the similarity search over spanning polytree. Thus, the less number of polytree levels, the less time the similarity search algorithm over a spanning polytree takes. The line chart in Figure 4.11 shows the number of polytree levels of the two procedures. The value of the black line (Opt_Find_Spanning_Polytree(G)) is always smaller or equal to the dotted line (Find_Spanning_Polytree(G)). Thus, for the number of polytree levels, Procedure 2 can extract a spanning polytree more efficiently than Procedure 1 with fewer polytree levels. The experimental results of Figure 4.11 agree with the procedure analysis in Section 4.5.1.

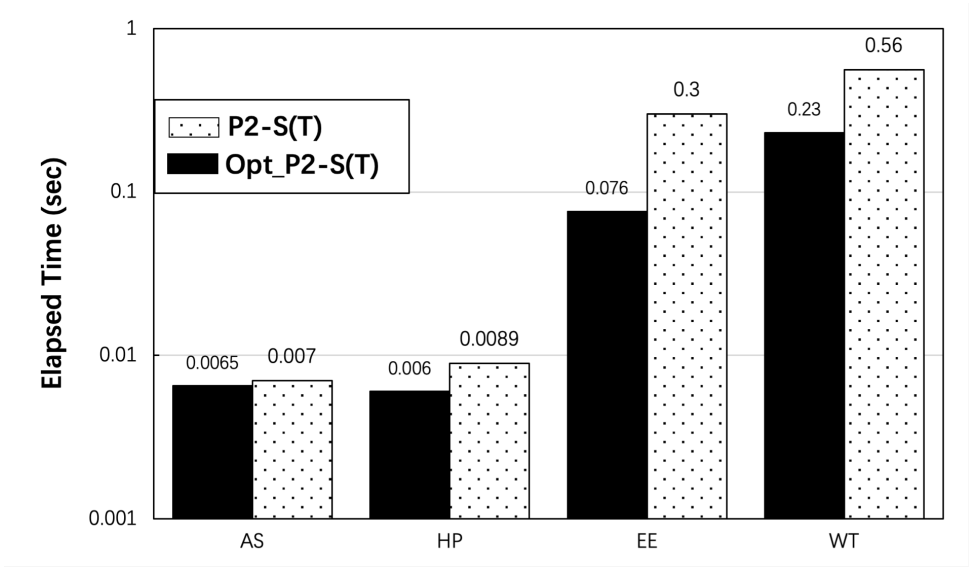


Figure 4.12: Time Efficiency Comparison of P2-$\mathbf{S}(T)$ and Opt_P2-$\mathbf{S}(T)$

Figure 4.12 shows the time efficiency of the similarity search and single-source similarity search on a spanning polytree algorithm over four datasets, AS, HP, EE and WT. P2-$\mathbf{S}(T)$ represents the second step of the F-CoSim algorithm, which is computing the CoSimRank scores of a spanning polytree. Opt_P2-$\mathbf{S}(T)$ is the optimization algorithm based on P2-$\mathbf{S}(T)$, and it is a single-source similarity search algorithm for a spanning polytree. Generating similarity scores of a spanning polytree is the second phase of F-CoSim. The results of the similarity search over a spanning polytree need to be added with $\Delta\mathbf{S}[:,Q]$; thus, the partial results of the similarity scores of a spanning polytree are required for the final results with respect to a query set. Figure 4.12 shows that the single source similarity search algorithm runs faster than the original similarity search algorithm. It is 3.9x faster than the original similarity search algorithm on EE.

Next, we evaluate the Opt_F-CoSim algorithm over three datasets. Opt_F-CoSim adopts the Opt_Find_Spanning_Polytree(G) procedure and the algorithm for a single-source similarity search on a spanning polytree, which replaces Find_Spanning_Polytree(G) and the original algorithm for a similarity search on a spanning polytree, F-CoSim.

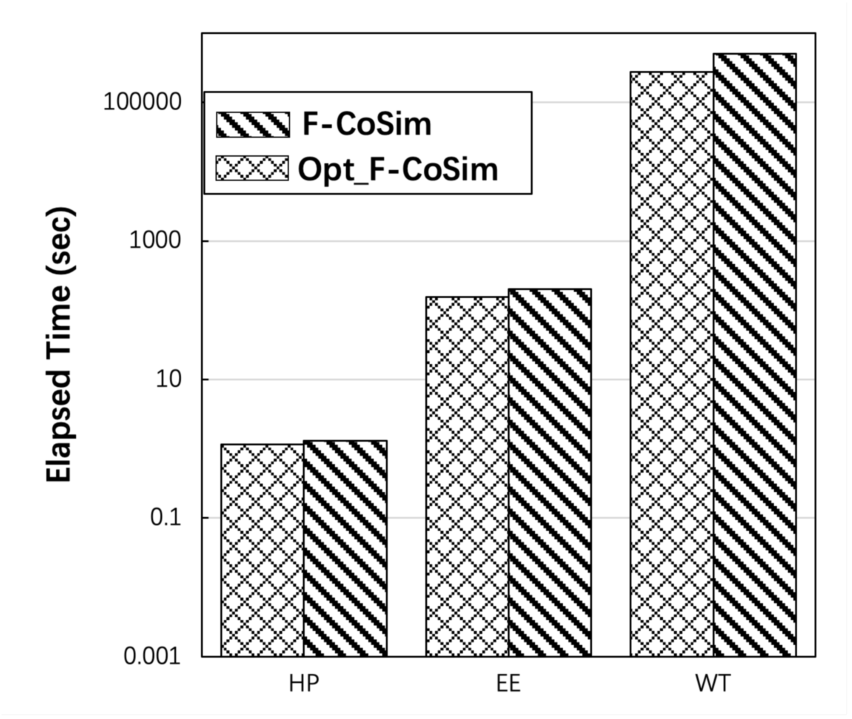


Figure 4.13: Time Efficiency Comparison of F-CoSim and Opt_F-CoSim

Figure 4.13 shows the time efficiency of F-CoSim and Opt_F-CoSim on three datasets (HP, EE and WT). We randomly select $|Q| = 500$ queries. The results in Figure 4.13 are the average time cost with respect to one query. The Opt_F-CoSim algorithm works faster than the F-CoSim algorithm on all three datasets. For example, Opt_F-CoSim algorithm is 1.2x faster than F-CoSim algorithm on EE.

Memory Efficiency. Figure 4.14 shows the memory efficiency of P2-$\mathbf{S}(T)$ and Opt_P2-$\mathbf{S}(T)$. P2-$\mathbf{S}(T)$ is the second phase of the F-CoSim algorithm, and Opt_P2-$\mathbf{S}(T)$ is the optimized version of P2-$\mathbf{S}(T)$. As illustrated in Figure 4.14, Opt_P2-$\mathbf{S}(T)$ consumes less memory space than P2-$\mathbf{S}(T)$ in case of all four datasets. For the computation of Opt_P2-$\mathbf{S}(T)$ in Procedure 2, the variables v and u are cleared after we get the CoSimRank scores of a spanning polytree in response to the query.

Figure 4.14 shows the memory efficiency of the F-CoSim and Opt_F-CoSim algorithms on three datasets. Opt_F-CoSim is approximately one order-of-magnitude more memory efficient than F-CoSim on HP, and shows a similar trend in its memory efficiency on the remaining

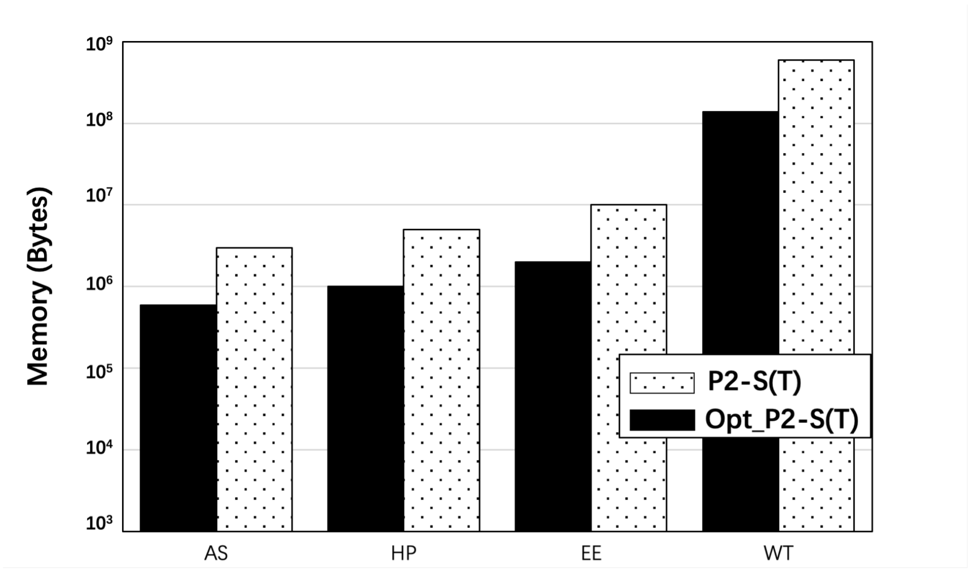


Figure 4.14: Memory Efficiency Comparison of P2-$\mathbf{S}(T)$ and Opt_P2-$\mathbf{S}(T)$

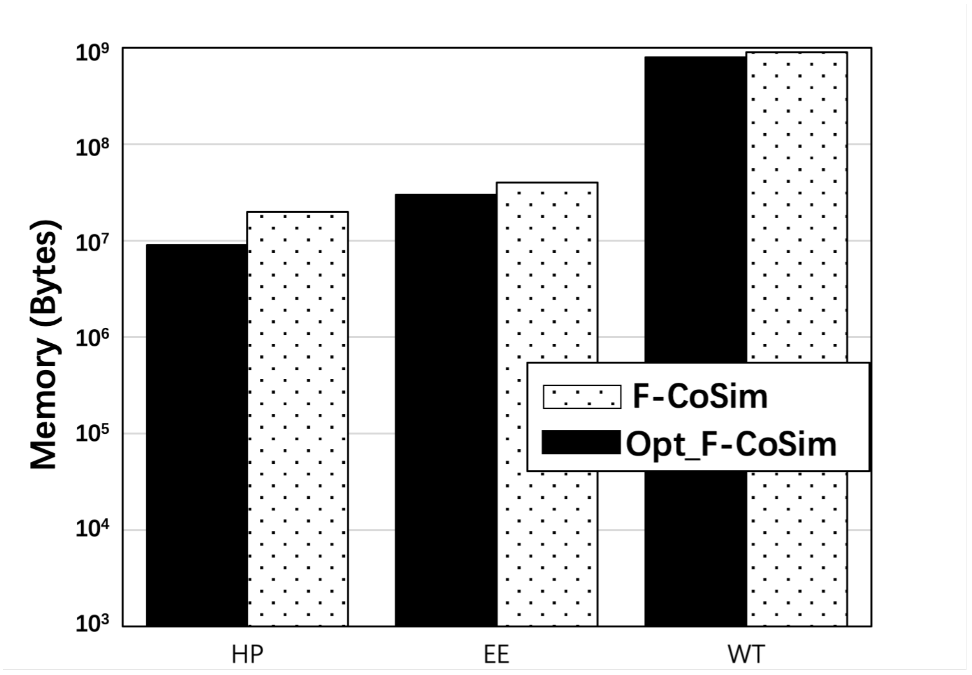


Figure 4.15: Memory Efficiency Comparison of F-CoSim and Opt_F-CoSim

datasets as well. Thus, Figure 4.14 shows that the two optimisation algorithms have improved the original F-CoSim algorithm.

Accuracy. We use the same accuracy evaluation method as D-CoSim, i.e., NDCG (Normalised Discounted Cumulative Gain) (Y. Wang et al., 2013), to evaluate the accuracy of the Opt_F-CoSim algorithm. The NDCG method has been explained in detail in Section 4.6.2.2.

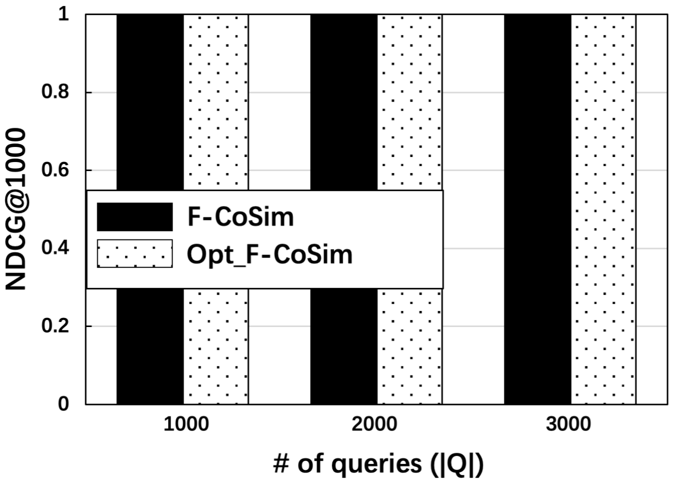


Figure 4.16: Accuracy of F-CoSim and Opt_F-CoSim

Here, we take the similar setting of NDCG as mentioned in Section 4.6.2.2. First, we randomly select different sizes of query sets ($|Q| = 1000$, $|Q| = 2000$ and $|Q| = 3000$). Then, we compare the results of F-CoSim and Opt_F-CoSim with the top $k = 1000$ CoSimRank ranking scores on AS. The trends on the remaining datasets are similar, and therefore have been omitted here. We recall the F-CoSim algorithm and show the results of its accuracy test in the figure. Figure 4.16 shows that all NDCGs of Opt_F-CoSim are equal to 1, which is the same as the F-CoSim algorithm. This means that the F-CoSim algorithm does not sacrifice any accuracy for speedup. The Opt_F-CoSim algorithm is the optimisation of F-CoSim with better time and memory efficiency, also keeps exact accuracy.

4.7 Related Work

There is a growing body of research on SimRank (the variant of CoSimRank) on static graphs (Yu, Zhang, Lin, Zhang, & Le, 2012b; Sarlós et al., 2006; Nguyen, Tomeo, Di Noia, & Di Sciascio, 2015; Fogaras & Rácz, 2005; Tian & Xiao, 2016; Kusumoto et al., 2014; Fujiwara et al., 2013; C. Li et al., 2010; Lizorkin, Velikhov, Grinev, & Turdakov, 2010; Jeh & Widom, 2002). The optimisation techniques suggested by these studies can be classified into three broad categories.

The first one is Monte Carlo sampling (Fogaras & Rácz, 2005; Sarlós et al., 2006; Tian & Xiao, 2016; Kusumoto et al., 2014). (Fogaras & Rácz, 2005) propose a quick similarity search algorithm , PSimRank, to retrieve a single pair SimRank score of two given vertices using the Monte Carlo simulation. The algorithm given by (Fogaras & Rácz, 2005) extended to multi-step neighborhoods with better theoretical characteristics and the Jaccard coefficient. (Sarlós et al., 2006) achieved unrestricted personalization by combining rounding and randomized sketching techniques in the dynamic programming of (Jeh & Widom, 2003). Furthermore, the algorithm proposed by (Sarlós et al., 2006) improved disk usage and communication complexity bounds of (Fogaras & Rácz, 2005). (Kusumoto et al., 2014) proposed a "linear" recursive framework that can efficiently compute a single-pair SimRank score the Monte Carlo simulation. The algorithm proposed by (Kusumoto et al., 2014) has a similar goal as (Fogaras & Rácz, 2005), but (Kusumoto et al., 2014) algorithm has better accuracy and memory efficiency than (Fogaras & Rácz, 2005). (Tian & Xiao, 2016) proposed an efficient index structure for SimRank, called SLING. The time complexity of SLING for single pair and single-source SimRank scores are $O(1/\epsilon)$ and $O(n/\epsilon)$ respectively, which is faster than PSimRank.

The second category is matrix-based methods (Fujiwara et al., 2013; C. Li et al., 2010). (C. Li et al., 2010) proposed a non-iterative form for the SimRank algorithm by using the Kronecker product and vectorization operators. To speed up the non-iterative SimRank algorithm, (C. Li et al., 2010) also developed approximate SimRank computation algorithms that are probably efficient with accuracy sacrifice. (Fujiwara et al., 2013) proposed an efficient top-k approximate algorithm, SimMat, based on the Sylvester equation to retrieve the SimRank scores of a given query. SimMat can also identify nodes whose SimRank scores are higher than the threshold efficiently.

The third category is iterative schemes (Yu, Zhang, et al., 2012b; Lizorkin et al., 2010; Jeh & Widom, 2002). The algorithm by (Jeh & Widom, 2002) is the most fundamental iterative similarity search algorithm over static graphs, and is called SimRank. SimRank has been successfully used in many practical applications, such as link prediction, web mining, and collaborative tagging analysis. The details of SimRank can be found in Section 3.2.1. However, although SimRank is known as a converge algorithm, when implemented in realistic applications, SimRank performs a finite number of iterative similarity searches. There may be a potential difference in the final SimRank scores of a practical application and its theoretical similarity scores. To fix this gap, (Lizorkin et al., 2010) presented a technique to estimate the accuracy of iterative SimRank computing. This technique can retrieve the iteration numbers of SimRank's computation to achieve the required accuracy. (Lizorkin et al., 2010) also provided a systematic improvement in order to reduce the computation complexity of SimRank. Unlike the technique proposed by (Lizorkin et al., 2010) to achieve a speedup using a partial sum function for amortization. (Yu, Zhang, et al., 2012b) proposed the SimFusion+ algorithm based on the notion of the UniFied Adjacency Matrix. SimFusion+ improves the time efficiency to $O(km)$ and the memory efficiency to $O(kn)$, which is more efficient than (Lizorkin et al., 2010).

Among the above existing works, the sampling approach, SLING (Tian & Xiao, 2016), is the best-of-breed SimRank algorithm on static graphs. However, their techniques, if applied to CoSimRank, are not fast, as the performance gain of SLING relies on aggregating only the *first* meeting time of two coalescing walks, as opposed to CoSimRank, which aggregates *all* their meeting times.

4.8 Conclusion

In this chapter, we discuss the issues that are missing in CoSimRank, such as computation efficiency and scalability, with an intention to facilitate a broader application of CoSimRank. To address these issues, we apply D-CoSim to static graphs by proposing F-CoSim to speed up large-scale static CoSimRank retrieval. On static graphs, our quick and accurate algorithm, F-CoSim, greatly speeds up CoSimRank retrieval based on three ideas: Given graph G, we (a) find a "spanning polytree" T of G; (b) design an efficient algorithm to retrieve CoSimRank scores $\mathbf{S}(T)$ over T; and (c) apply D-CoSim to evaluate the changes to $\mathbf{S}(T)$ in response to delta graph $(G \ominus T)$; and (d) propose a technique for faster extraction of a spanning polytree from a graph with fewer levels number in order to speed up F-CoSim. Since F-CoSim is a single-source similarity search algorithm, changing the second phase of F-CoSim from all nodes pairs similarity search $\mathbf{S}(T)$ over T to single-source similarity search $\mathbf{S}(T)[:, q]$ over T will speed up the F-CoSim algorithm. Based on these two optimization techniques, we propose the Opt_F-CoSim algorithm. Furthermore, inspired by parallel computing, we also propose the

F-CoSim_Para algorithm for more efficient computation of F-CoSim. Our empirical studies on various real datasets demonstrate that (a) F-CoSim outperforms state-of-the-art approaches on static graphs, with a speedup of up to 9.8 times; (b) F-CoSim retains comparable linear memory, and scale on million-node graphs, with no compromise of accuracy for the speedup. (c) The results after implementing Opt_Find_Spanning_Polytree on six realistic datasets show that Opt_Find_Spanning_Polytree is more efficient than Find_Spanning_Polytree. (d) Opt_F-CoSim outperforms F-CoSim on static graphs with a speedup of approximately 1.2 times (time efficiency), and approximately one order-of-magnitude memory efficiency.

For future works, we will extend our CoSimRank scheme to decentralised environments for picture evaluation and patient tracking systems, for example, picture semantic similarity search (M. Zhang et al., 2021), image-rich web sets similarity integration (Muni Manasa & Farooq, n.d.), and virus spreading prediction (Colliri & Zhao, 2020).

5 An Axiomatic Role Similarity Search Over Networks

Networks are ubiquitous in our daily life. For instance, let us think of social networks (Everett, 1985). The social network for each of us is made up of interactions and connections between people. Humans are social animals, just like what John Donne's *No Man Is An Island* says:

> "No man is an island entire of itself; every man is a piece of the continent, a part of the main; if a clod be washed away by the sea, Europe is the less, as well as if a promontory were, as well as any manor of thy friends or of thine own were; any man's death diminishes me, because I am involved in mankind. And therefore never send to know for whom the bell tolls; it tolls for thee." - John Donn *No Man Is An Island*

With the continuous development of electronic communication technology in the past decade, people's social networks have been greatly expanded. People do not need to communicate face-to-face or be introduced by a friend to establish contacts for the social network. In recent years, people have been using mobile phones, computers and other devices to establish contact with others through chat software, online shopping sites, social medial sites or apps. The development of networks leads to exponential growth of people's social networks. Therefore, the similarity detection algorithm for the network structure is urgently needed. The results of similarity detection can greatly improve people's quality of life, such as big data research, scenario prediction etc.

In the previous two chapters, we presented how the CoSimRank scores can be calculated efficiently from a large static or dynamic graph. CoSimRank (or any other SimRank-liked similarity measures) is mainly used to measure if two single nodes in graphs are similar. Apart from SimRank-liked similarity measures, there is another kind of very popular graph-theoretic similarity measures, which is the role similarity. Role similarity aims to address the role (automorphic) equivalence of pairwise similarity. This chapter presents an innovative role similarity search algorithm over networks.

5.1 Introduction

Graph structures (networks) commonly exist in our daily life, such as protein and neural structure in the field of biological science (Rao, Devi, Kaladhar, Sridhar, & Rao, 2009), Amazon and Google scholar recommendation system in Internet surfing (Shahabi, Banaei-Kashani, Chen, & McLeod, 2001) and Facebook network and family relationship structure graph in social life (da Silva Villaca, de Paula, Pasquini, & Magalhaes, 2013). Thus, the research on extracting useful information based on topology structure becomes extremely important.

Role similarity analysis (Everett, 1985) is an essential method for analysing real complex graph structures, especially for a social network, as it can accurately capture structural information from a network. The most basic principle for role similarity search is that two nodes have a similar role only if they interact with similar objects. As presented in previous chapters, on the basis of this recursive node similarity search principle, many research works have been conducted on similarity search over networks, such as Personalised PageRank (Haveliwala, 2003), SimRank (Jeh & Widom, 2002) etc. These similarity search scores and algorithms can capture the structural equivalence from graphs. However, they cannot generate the role similarity scores based on the graph topology.

There are two significant challenges (Lee, 2012) we have to face when we transfer our study focus from the structural equality-enforcing similarity algorithms to the role similarity algorithms. The first is how to define the roles in the graph structure, and the second is how to conduct the roles' similarity search based on graph topology. For the definition of "role" based on graphs, the set of relationships between an individual and others is referred to as a role. For the definition of role similarity search over networks, given a graph $G = (\mathbf{V}, \mathbf{E})$, for a node v, the set of all edges incident to it, i.e. v $([u_1, u_2, \cdots, u_{deg_v}] \to v) \in \mathbf{E}$, is the role of v in graph theory. Intuitively, for example, if two people share the same relationships in a social network, they play the same roles.

The method of defining the role similarity is to detective the role equivalence. In the study of the role similarity, there are four types of role equivalence, including *structural equivalence*, *automorphic equivalence*, *equitable partition* and *regular equivalence*. In this chapter, we have

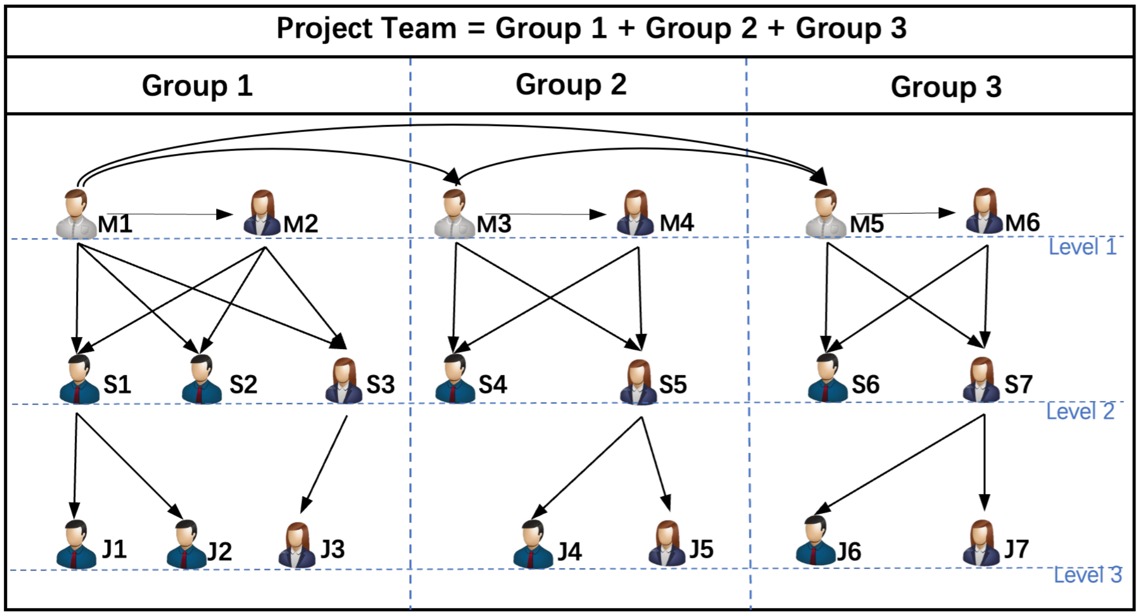


Figure 5.1: A Social Network of A Project Studio With Three Levels (Level 1: project manager/associate project manager; Level 2: senior staffs; Level 3: junior staffs).

mainly considered the automorphic equivalence for the role similarity search.

Next, we have used an example to illustrate the application, significance of role and role similarity detection based on graph topology in our daily life.

Figure 5.1 is a typical social network of a project team. The nodes in the figure represent the employees who participate in the project, and the edges are the interactions and connections between employees. The complete project team is divided into three groups according to the different tasks being taken. The members of each group are separated into three job levels in consonance with their different positions, which are project managers or associate project managers (level 1), senior employees (level 2), and junior staffs (level 3). A member's position in this project is considered as his/her role. A role similarity detection algorithm aims to quickly and precisely find out the nodes which play the similar roles as a given query node in the social network, and This is an algorithm that takes a node as the query and generates the role similarity scores between all nodes in the graph and the query node, which is called a `single-source` role similarity searching algorithm. For example, taking the node $J3$ as the query, $[J4.J5, J6, J7, J1, J2]$ should have higher role similarity scores with $J3$ comparing with other nodes like $S1$ or $M1$ since they have the similar role in the project as $J3$ (i.e., junior staffs).

The following are some practical examples of role similarity detection. They demonstrate the necessity for more precise role similarity search algorithms to handle topology-based assessments over large networks.

Application 1 (Co-authorship Network).

A co-authorship network is a social network where the authors who participated in one or more publication are linked to each other through an indirect path. The role similarity detection in the co-authorship network is an important application. In the co-authorship network, each author can be defined as a node, and an edge is added between authors if they have worked together on a publication. The RoleSim algorithm (Rothe & Schütze, 2014) can be applied over a co-authorship network. According to the definition of RoleSim, "an author's role depends recursively on the number of connections to other authors and the roles of those others". However, the results of the RoleSim algorithm can be improved since the algorithm only captures partial information from the neighbour nodes in a graph.

Application 2 (Recommendation Systems of an eCommerce Website). The continuous development of internet has changed people's lifestyles to a large extent. Over the years, the user numbers of major online shopping sites, such as Amazon and eBay, have increased exponentially. At the same time, the types and quantities of online selling products are also increasing substantially. Targeted item recommendations can save users' shopping time while promoting product sales. The analysis of the users' role similarity helps in refining the recommendation system (Diao, Wang, Alsarra, Yen, & Bastani, 2019). In an eCommerce website, each registered user can be defined as a node; an edge can be added between users if they have purchased the same product. The role similarity search algorithm can divide users into different equivalence groups and recommend products to a user if other users in the same group have purchased them.

5.1.1 Motivation

The pioneering research on role analysis on networks was conducted by Jin (Lee, 2012). Jin's role similarity search algorithm, known as RoleSim, computed each similarity score from the average of maximum matching values of in-neighbours. While having addressed the automorphic equivalence of pairwise similarity successfully, Jin's RoleSim algorithm has several limitations.

Limitation 1 (Accuracy). The role similarity score of RoleSim is computed by calculating the maximum matching (Gabow, Kaplan, & Tarjan, 2001) of neighbours' similarity score matrix which is limited by Eq. 5.3. In RoleSim, the maximum matching problem is solved by using solutions of the "assignment problem" (Burkard, Dell'Amico, & Martello, 2012). The assignment problem is a well-known combinatorial optimisation problem. The problem instance has some workers and jobs. Any worker can be assigned any job but with different cost. Each worker can only do one job, and each job can only be done by one worker. It is required that the workers be assigned different jobs in such a way that the total cost of the assignment is minimised.

When the maximum matching algorithm is applied in the role similarity search algorithm, some useful connection information in the graph is ignored. There are two primary causes for this potential information loss. First, the assignment problem requires that the number of workers and jobs be the same, meaning the assignment matrix has to be a square matrix. For this reason, the RoleSim search algorithm requires the in-neighbour matching matrix be a square matrix as well. When the in-neighbour score matrix is not a square, i.e. the number of rows differs from the number of columns, the RoleSim algorithm discards those in-neighbour nodes representing the extra rows/columns in the matrix from the matching selection process. Furthermore, though an in-neighbour is included in the matching selection process, some of its similarity scores are ignored, as the RoleSim algorithm requires only one value to be extracted from each row and each column of the in-neighbour score matrix.

Therefore, these two factors can compromise on the accuracy of the RoleSim algorithm and may lead to unclear role classification. To bridge this gap, we proposed a novel role-based similarity search algorithm FaRS on networks. Example 16 and Figure 5.3 have been taken as examples to illustrate the accuracy limitations of the RoleSim algorithm.

Example 16. *Given a graph $G = (\mathbf{V}, \mathbf{E})$ and two nodes $(u, v) \in V$, the in-neighbour set of the node u is $I(u) = a, b, c$, and the in-neighbour set of v is $I(v) = d, e, f, g, h$ in the graph. The maximum matching result of the in-neighbour matrix of node-pair (u, v) is shown in Figure 5.2, where $\mathcal{M}_i^k$ denotes the top k maximum matching result on i^{th} row. As evident from Figure 5.2, the maximum matching results of the in-neighbour matrix, which are cycled by a solid square, are as follows: $\mathcal{M}^1(u, v) = \sum_{i \in \mathcal{I}(u)} \mathcal{M}_i^1 = \mathcal{M}_a^1 + \mathcal{M}_b^1 + \mathcal{M}_c^1$.*

$\mathcal{M}^1(u, v)$ is the first-order maximum matching result of node pair (u, v) in-neighbour matrix produced by RoleSim. The result of $\mathcal{M}^1(u, v)$ shows that it only captures information (solid square) from column (d, e, f) (green area), and the information in column (g, h) is ignored (red

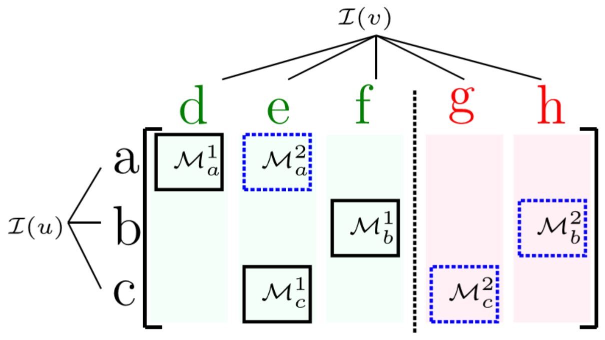


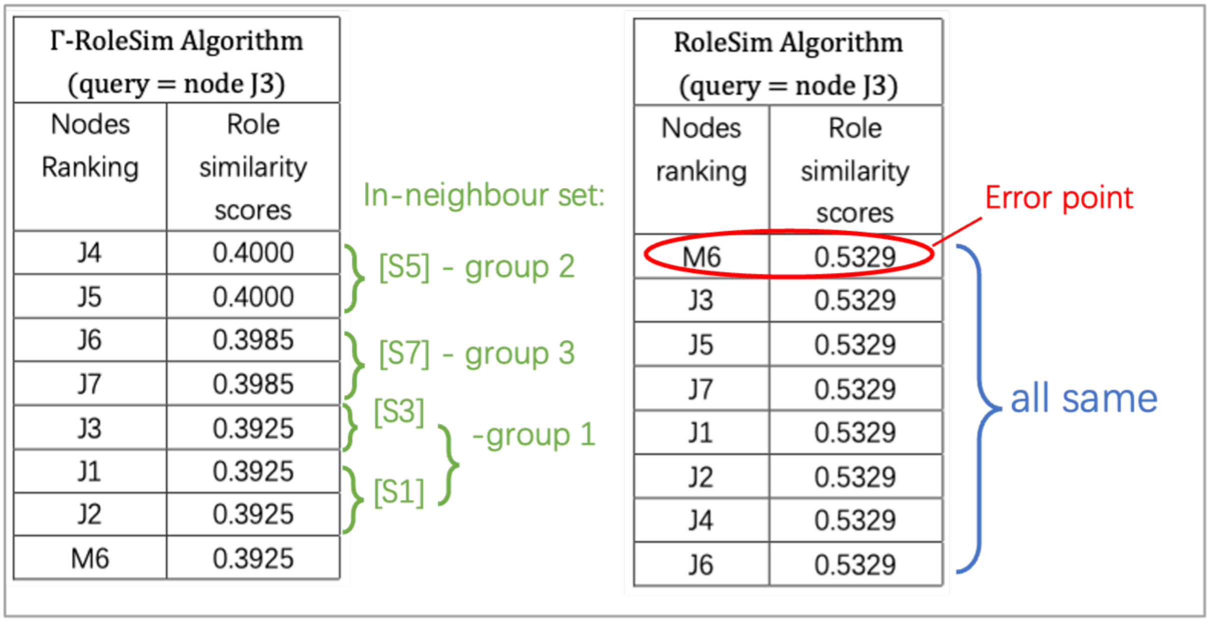
Figure 5.2: In-Neighbour Matrix of Node-Pair (u, v)

area). The particular reason for RoleSim *missing the information in the red area is that the in-degree of node u is smaller than the in-degree of node v, and maximum matching only captures the smaller in-degree information from the in-neighbour matrix. It means that* RoleSim *can only capture the* $mindeg^-(u,v) = min(deg_u^-, deg_v^-)$ *row/column information from the node-pair's in-neighbour matrix. Especially, when* $deg_u^- \neq deg_v^-$*,*
the $(maxdeg^-(u,v) - mindeg^-(u,v))$ *nodes' in-neighbours-related information will be ignored, which leads to accuracy issues.*

Recalling the example in Figure 5.1, we used it to show that the accuracy of the RoleSim algorithm needs to be improved. We take node $J3$ as a query in the project team social network. The role similarity search results produce by using the RoleSim algorithm and our proposed FaRS algorithm are shown in Figure 5.3. The details of FaRS have been presented later in this chapter. In Figure 5.3, the table on the right shows the role similarity results produced by the RoleSim algorithm in response to the query $J3$, and the left table is the results of the FaRS algorithm with respect to query $J3$. In the right table (RoleSim algorithm), the role similarity search scores between the node $J3$ and $[M6, J3, J5, J7, J1, J2, J4, J6]$ are the same, which actually implies that the node $J3$ and the nodes in the set $[M6, J3, J5, J7, J1, J2, J4, J6]$ all belong to the same role classification. However, $[J3, J5, J7, J1, J2, J4, J6]$ certainly belong to the junior staff classification (level 3), but $M6$ belongs to the manager classification (level 1). Therefore, the results of RoleSim may exit error points that need to be fixed.

The results produced by our newly proposed algorithm FaRS (the right table in Figure 5.3) can reflect this fact correctly.

Another accuracy problem of the RoleSim algorithm is that it cannot recognise the structural



Γ-RoleSim Algorithm (query = node J3)	
Nodes Ranking	Role similarity scores
J4	0.4000
J5	0.4000
J6	0.3985
J7	0.3985
J3	0.3925
J1	0.3925
J2	0.3925
M6	0.3925

RoleSim Algorithm (query = node J3)	
Nodes ranking	Role similarity scores
M6	0.5329
J3	0.5329
J5	0.5329
J7	0.5329
J1	0.5329
J2	0.5329
J4	0.5329
J6	0.5329

In-neighbour set:
[S5] - group 2
[S7] - group 3
[S3]
-group 1
[S1]
Error point
all same

Figure 5.3: The role similarity search results of RoleSim and FaRS

equivalence (group classification in Figure 5.1). In the left table (FaRS) in Figure 5.3, unlike RoleSim where $J1$ to $J7$ have the same role similarity search scores, the scores produced by FaRS are different and therefore, the similarity levels can be ordered and ranked. Among them, the role similarity scores between node $J3$ and nodes $[J4, J5]$ are 0.4, and both $[J4, J5]$ belong to group 2. Similarly, we can recognise $[J6, J7]$ and $[J3, J1, J2]$ belonging to group 1 and group 3 respectively. Note that even $J3$ and $[J1, J2]$ have different the in-neighbour (the in-neighbour of node $J3$ is node $S3$ and the in-neighbour of $[J1, J2]$ is node $S1$), but because both nodes $S3$ and $S1$ belong to group 1, $J3$ and $[J1, J2]$ are structurally more closed. FaRS can recognize this fact, which can be seen in the results of FaRS ($FaRS(J1, J3) = FaRS(J2, J3) = FaRS(J3, J3)$). This example fully illustrates that FaRS not only corrects the RoleSim error point and improves the accuracy of automorphic equivalence, but also yields structural equivalence, which is completely ignored by RoleSim.

Limitation 2 (Computational Efficiency).

The definition of single-source search is to target a node of a graph and retrieve the similarity relationship between this node and all other nodes in this graph. For example, in the case of the Amazon recommendation system (Linden et al., 2001), users search a product on Amazon, and the product can be taken as a query, and then the system shows the products most relevant to this query in the recommendation list. Single-source search plays an increasingly critical role in our daily life; thus, an efficient single-source role similarity search algorithm over graphs needs to be developed urgently. However, the RoleSim algorithm is quite time-consuming when computing similarity scores between a random query with all objects in graphs, as it has futile computations. At the same time, the computation of repeated maximum matching algorithm is time-consuming. Therefore, significantly reducing the call number of maximum matching algorithm is a vital acceleration method.

The FaRS algorithm which we propose in this chapter conducts a single-source role similarity search based on graph topology, which can efficiently role similarity search scores over large graphs.

Limitation 3 (Dynamic Graph).

As discussed in Chapter 3, in many real applications, the structure of graphs evolves constantly. Taking the Facebook social network as an example, the number of users (nodes of the graph) and the relationship between any two users (edges of the graph) are changing all the time. Thus, an efficient and accurate role similarity search algorithm over dynamic graphs is essential to our life. However, there exists little research on role similarity search over dynamic graphs. The existing role similarity search algorithms like RoleSim face the same performance challenge as CoSimRank (Chapter 3) when they are applied over dynamic graphs. The reason is that when a graph undergoes changes, even slight ones such as one node or one edge being added into the graph, the RoleSim algorithm needs to recompute the similarity scores of all node pairs in the graphs, which is extremely time-consuming. This prompts us to find an accurate and efficient algorithm to compute the role similarity scores over dynamic graphs.

To address these limitations of the existing role similarity search algorithms, this chapter mainly solves the following questions.

Problem (Accurate Single-Source Role Similarity Search Based On Graph Topology).

Given: a connected graph $G = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ represents the set of nodes and $\mathbf{E}$ denotes the set of edges in graph G and a query $q \in \mathbf{V}$.

Retrieve: the role similarity scores with respect to q on graph G quickly and accurately.

To improve the accuracy of the RoleSim algorithm, FaRS, an innovative role similarity search algorithm has been proposed in this chapter. Here, Γ represents the maximum order of the maximum matching applied in the role similarity search algorithm. $0 < \Gamma < n$, n is the number of nodes in graphs. We computed the similarity scores between a random query q with all other objects in graph G. It was found that the FaRS algorithm can capture more useful information from the in-neighbours similarity score matrix; thus, the role of each node can be reflected more accurately in the network. $\Gamma = 2$ is an exceptional case of FaRS algorithm, and we called the FaRS as Sec-RoleSim when we set $\Gamma = 2$. The details of Sec-RoleSim have been illustrated in Section 5.3.2.

Then, we proposed Opt_FaRS – an optimisation algorithm of FaRS with two acceleration techniques, which are the P-speedup approach and the Out-speedup approach. These acceleration technologies can greatly reduce the number of calls of the maximum matching algorithm. Therefore, when the single-source role similarity values of a graph are calculated, a part of the results of the maximum matching algorithm for some node pairs can be shared with other node pairs, and our acceleration technology extracts and reuses these interim results to reduce repetitive calculations.

Finally, since FaRS is a single-source role similarity search algorithm over graphs, we found that it can be applied on dynamic graphs.

FaRS and Opt_FaRS have the following distinguishing qualities based on the aforementioned concepts.

- **Accurate.** FaRS and Opt_FaRS have more accurate role classification results than the best-known competitors.
- **Fast** FaRS can efficiently compute single-source role similarity search results over graphs. Opt_FaRS, as an optimisation of FaRS, can further improve the performance, as it can extract shared information to greatly reduce the repetitive computation.
- **Dynamic.** FaRS and Opt_FaRS are single-source role similarity search algorithm over networks, and they can be applied over dynamic graphs with pre-processes.
- **Index-free.** Our schemes do not need extra disk space for storing indexing results.

To sum up, FaRS and Opt_FaRS enable the more accurate and faster handling of a wide range of role-based similarity search applications (Rothe & Schütze, 2014; Shao et al., 2019; Lin, Lyu, & King, 2012).

5.1.2 Chapter Outlines

The rest of this chapter is organised as follows:

Chapter 5.2 In this section, we recall the current RoleSim algorithm (Rothe & Schütze, 2014) and analyse the limitations of the algorithm while applying them over graphs.

Chapter 5.3 We propose an accurate role similarity search algorithm FaRS based on graph topology in this section. We also prove the convergence, uniqueness, symmetry, boundedness and triangular inequality of the FaRS algorithm.

Chapter 5.4 In this section, we propose the Opt_FaRS algorithm, which optimised FaRS. Opt_FaRS consists of two accelerating components:

  tracking path extraction we record the tracking path starting from the query node.

  pre-computation we use the p-speedup and out-speedup approach to efficiently retrieve the candidate pool of each level.

Chapter 5.5 We implement the Opt_FaRS algorithm over dynamic graphs. It comprises the following main steps: (i) chunking the updated part of the graphs into groups, with each group combining all the updated ends with the same node; (ii) updating the adjacency matrix of the dynamic graphs in response to each updated chunk; (iii) implementing the Opt_FaRS algorithm with the new adjacency matrix.

Chapter 5.6 We conduct several experiments on real datasets to demonstrate that our FaRS and Opt_FaRS algorithms are steadily becoming more accurate than the two state-of-the-art CoSimRank competitors (CSR (Rothe & Schütze, 2014) and RS (Rothe & Schütze, 2014)). The experimental results also show that our algorithm Opt_FaRS does not scarify accuracy for speeding up.

5.2 Preliminary

In this section, we define some notations used in this paper, and revisit formulae and properties of the RoleSim algorithm. RoleSim, investigated by (Rothe & Schütze, 2014), is an attractive node-pair role similarity search algorithm over networks (graphs). The RoleSim algorithm was founded on the recursive philosophy that "two nodes have the same role if they interact with equivalent sets of neighbours". Thus, the algorithm retrieves the role similarity scores and performs role classification based on how node pairs interact with others.

In general, there are four types of equivalence in the analysis of network similarity: structure equivalence, automorphic equivalence, exact colouration and regular equivalence (Rothe & Schütze, 2014). Among them, the most fundamental equivalence for role similarity is automorphic equivalence. RoleSim can produce the real-valued role similarity measure which confirmed automorphic equivalence. Before revisiting RoleSim in details, the basic intuition and several notations used in this paper are introduced first.

Given is a connected graph $G = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ represents the set of nodes in G and $\mathbf{E}$ denotes the edges in G, respectively. The neighbour definition of node $v(v \in \mathbf{V})$ can be found in Definition 3 (Chapter 3). In Definition 3, $\mathcal{I}(v)$ and $\mathcal{O}(v)$ are the in-neighbour set and the out-neighbour set of the node v in the graph, respectively. deg_v^-/deg_v^+ is the in-degree/out-degree of v in the graph, respectively. Here, we defined $mindeg^-(u,v)$ as the smaller in-degrees of node u and node v, mathematically, $mindeg^-(u,v) = min(deg_u^-, deg_v^-)$. We also defined $maxdeg^-(u,v) = max(deg_u^-, deg_v^-)$. $\mathcal{M}(i,j)$ is the maximum matching (Yannakakis & Gavril, 1980) result of the in-neighbour matrix of node pair (i,j). Take the nodes pair $(S1, J1)$ in Figure 5.1 as an example: $deg_{S1}^- = 2$, $mindeg^-(S1, J1) = min(2,1) = 1$, and $maxdeg^-(S1, J1) = max(2,1) = 2$. The definition of maximum matching is shown as follows.

Definition 7 (**Maximum Matching**). *Given two node sets* $\mathbf{u}$ *and* $\mathbf{v}$ *and the value of each node-pair* $(i, j | i \in \mathbf{u}, j \in \mathbf{v})$ *is* $c_{i,j}$*, maximum matching can be stated as the maximum total value:*

$$Maxm(\mathbf{u}, \mathbf{v}) = \sum_{i=1}^{|\mathbf{u}|} \sum_{j=1}^{|\mathbf{v}|} c_{i,j} x_{i,j} \tag{5.1}$$

where $x_{i,j} = \begin{cases} 1 & if\ i^{th}\ node\ of\ \mathbf{u}\ is\ assigned\ to\ the\ j^{th}\ node\ of\ \mathbf{v}; \\ 0 & if\ i^{th}\ node\ of\ \mathbf{u}\ is\ not\ assigned\ to\ the\ j^{th}\ node\ of\ \mathbf{v}. \end{cases}$

at the same time, Eq (5.1) *subject to the constrain:*

- $\sum_{i=1}^{|\mathbf{u}|} x_{i,j} = 1 (j = 1, 2, \cdots, |\mathbf{v}|)$*, which means for each node in* $\mathbf{v}$*, only* j^{th} *node assigned to* i^{th} *node in* $\mathbf{u}$.

- $\sum_{j=1}^{|\mathbf{v}|} x_{i,j} = 1 (i = 1, 2, \cdots, |\mathbf{u}|)$*, which means for each node in* $\mathbf{u}$*, only* i^{th} *node assigned to*

j^{th} node in $\mathbf{u}$.

here $|\mathbf{u}|$ and $|\mathbf{v}|$ represent the number of nodes in node sets $\mathbf{u}$ and $\mathbf{v}$, respectively. In this chapter, $\mathcal{M}(i,j) = Maxm(\mathcal{I}(i), \mathcal{I}(j))$.

RoleSim is founded on the concept of maximal matching of neighbor similarity, where the similarity between nodes are recursively defined as the average similarity of the maximum weight matching between their neighbours. The maximum matching algorithm (Yannakakis & Gavril, 1980) forms the basis of the job assignment problem, which is a combinatorial optimisation algorithm to solve the problem in polynomial time. In this paper, $\mathcal{M}(i,j)$ denotes a maximum matching result of the in-neighbour matrix of node pair (i,j), i.e $\mathcal{M}(i,j) = \mathcal{M}(I(i), I(j))$.

The abbreviation rs represents the role similarity score of two nodes in the RoleSim algorithm. Mathematically, the formula of the RoleSim algorithm for retrieving the role similarity score of two nodes $(u,v) \in G$ can be written as follows:

$$rs(u,v) = (1-C) \max_{\mathcal{M}(j,i)} \frac{\sum_{(x,y)\in\mathcal{M}(j,i)} rs(x,y)}{deg_i^- + deg_j^- - |M(j,i)|} + C \tag{5.2}$$

where C is the decay factor $(0 < C < 1)$. $deg_i^- + deg_j^- - mindeg^-(i,j)$ equal to $maxdeg^-(u,v)$. The upper denominator in Eq. (5.2) is the maximum matching results of the in-neighbour matrix of node-pair (u,v). To prevent the formula from being divided by zero, the RoleSim score is designed as C when deg_i^- or deg_j^- equals 0. This is not like SimRank (Jeh & Widom, 2002) or CoSimRank (Rothe & Schütze, 2014), where if the in-degree of node u or node v is zero, the similarity score between the two nodes is defined as 0 in both algorithms.

The current algorithms compute the role similarity score of RoleSim iteratively as the following steps.

The first phase is the initialisation of the role similarity search scores matrix. The second phase is to calculate the role similarity score between the node pair $(u,v) \in \mathbf{V}$ at the k^{th} iteration with the role similarity scores of the $(k-1)^{th}$ iteration.

$$rs_k(u,v) = (1-C)\frac{\mathcal{M}_{k-1}(u,v)}{maxdeg^-(u,v)} + C \tag{5.3}$$

The symbol k denotes the iteration times. $\mathcal{M}_{k-1}(u,v)$ denotes the maximum matching results of the in-neihgbour matrix of node-pair (u,v) at the $(k-1)^{th}$ iteration. The in-neighbour matrix of node pair (u,v) is constituted by the $\mathcal{I}(u)^{th}$ row and the $\mathcal{I}(v)^{th}$ column of $\mathbf{rs}^{k-1}$. The second phase is repeated until the convergence of all node-pair role similarity scores.

The RoleSim algorithm has the following properties, which are stated in (Rothe & Schütze, 2014):

1. **Boundedness:** The similarity score $rs(*, *)$ always exists and is unique, and $C \leq rs(*, *) \leq 1$.
2. **Monotone Convergence:** The value of $rs_k(*, *)$ is the upper bound of $rs_K(*, *)$, *i.e.,* $rs_k(*, *) \geq rs_{k+1}(*, *)$.
3. **Convergence:** The result of RoleSim is converging to $rs(*, *)$, *i.e.,* when k approaches infinity, $lim_{k\to\infty} rs_k(*, *) = rs(*, *)$.
4. **Triangle inequality:** The RoleSim algorithm meets the requirement of the triangle inequality.

Later, we will prove that these properties are also held in our algorithm FaRS and Opt_FaRS (Section 5.3).

5.3 Proposed Schema

In this section, FaRS, a single-source role similarity search algorithm over graphs, has been presented. Compared with the RoleSim algorithm, FaRS is better at discovering and categorising the nodes of graphs. Here, Γ represents the maximum order of the maximum matching applied in the role similarity search algorithm. Since the different values of Γ in FaRS do not affect the properties of the FaRS algorithm presented in this thesis, we used a special case of FaRS – Sec-RoleSim, which was the FaRS role similarity search algorithm with $\Gamma = 2$, to demonstrate the process of FaRS. Furthermore, the proofs of various FaRS algorithm properties are illustrated by Sec-RoleSim to save space.

5.3.1 FaRS: A Role-Based Similarity Algorithm Over Graphs

This subsection details our accurate and efficient role similarity search algorithm, FaRS. As reviewed in the previous section, the well-known role similarity algorithm RoleSim retrieves node-pair role similarity scores by capturing the maximum matching of the node pair in-neighbour matrix. The maximum matching algorithm (Yannakakis & Gavril, 1980) forms the basis of the assignment problem, which is a combinatorial optimisation algorithm to solve the problem in polynomial time. The most fundamental principle of the assignment problem is that "there are u tasks and v staffs in a company, assigning works depends on the rule that each task only can be done by one staff and each staff only can do one task so that the totally cost of company is minimum". The maximum matching technique can be used in the RoleSim algorithm. It means that RoleSim can only capture the $mindeg^{-}(u,v) = min(deg_u^{-}, deg_v^{-})$ row/column information from the node-pair in-neighbour matrix. Especially, when $deg_i^{-} \neq deg_j^{-}$, the $(maxdeg^{-}(u,v) - mindeg^{-}(u,v))$ nodes' in-neighbours-related information will be ignored. This limitation of the RoleSim algorithm leads to accuracy issues.

To perform more accurate role-based similarity search, we propose FaRS algorithm. To distinguish the single-source role similarity scores in the equation of FaRS (Eqs. 5.4), here $\mathbf{RS}^{\Gamma}(set_A, q)$ represents single query role similarity scores between $set_A \in \mathbf{V}$ and query q, $\mathbf{RS}^{\Gamma}(set_A, set_B)$ is role similarity scores matrix between set_A and $set_B \in \mathbf{V}$, and $RS^{\Gamma}(u,v)$ is the role similarity score of node pair (u,v), and Γ represents the number of the best matching in maximum matching applied (i.e, top Γ maximum matching pairs are selected) in the role similarity search algorithm.

Theorem 15 (**FaRS**)**.** *Given a connected graph G, and a query $q \in V$, the FaRS role similarity scores with respect to q are defined as:*

$$\begin{aligned} \mathbf{RS}^{\Gamma}(:,q) = (1-C)(\max_{\mathcal{M}^1(:,q)} \textstyle\sum_{(x,y)\in\mathcal{M}^1(:,q)} RS^{\Gamma}(x,y) + \lambda \\ \max_{\mathcal{M}^2(:,q)} \textstyle\sum_{(x,y)\in\mathcal{M}^2(:,q)} RS^{\Gamma}(x,y) + \cdots + \lambda^{(\Gamma-1)} \cdot \max_{\mathcal{M}^{\Gamma}(:,q)} \textstyle\sum_{(x,y)\in\mathcal{M}^{\Gamma}(:,q)} RS^{\Gamma}(x,y)) \\ \oslash(1+\lambda+\cdots+\lambda^{(\Gamma-1)})(\mathbf{deg}_{i=1:n}^{-} + \mathbf{deg}_q^{-} - mindeg^{-}(:,q))) + C \end{aligned} \tag{5.4}$$

where $M^{\Gamma}(:,q)$ is the top Γ^{th} order maximum matching of the in-neighbour matrix of nodes $(i=1:n)$ and query q. λ is the normalisation coefficient $(0 \leq \lambda \leq 1)$. $\oslash$ denotes the division of the values of the corresponding positions of two vectors. $\mathbf{deg}_{i=1:n}^{-}$ is a vector whose values are in-degree of node $i(i \in \mathbf{V})$. $\mathbf{deg}_q^{-}$ is a vector, and all its values are in-degree of node q. $|\mathcal{M}(:,q)|$ is a vector, and the values of this vector are the minimum value of node-pair $[(i,q)|i \in \mathbf{V}]$.

Theorem 15 shows that the FaRS role similarity algorithm captures Γ^{th} order maximum matching values from a node pair in-neighbour matrix, which can guarantee more accuracy than RoleSim. To prevent dividing zero from numerator in Eqs. 5.4, some special cases are included in the equation:

$$\begin{cases} RS^{\Gamma}(u,q) = C \qquad deg_u^- = 0 \quad or \quad deg_q^- = 0 \\ \\ \mathbf{RS}^{\Gamma}(:,q) = \mathbf{C}_{n\times 1} \qquad deg_q^- = 0 \end{cases}$$

Next, Example 17 is used to illustrate the difference between the RoleSim algorithm and the FaRS algorithm.

Example 17. *Recall Figure 5.2, when we use the FaRS algorithm to retrieve the role similarity search scores between the node pair (u, v), we first set $\Gamma = 2$. Setting $\Gamma = 2$ actually means that the role similarity scores should be calculated by the top 2 maximum matching values. In Figure 5.2, the first order of maximum matching values (the largest matching values) are marked by a solid square. The second-order (the second largest) maximum matching of the in-neighbour matrix are nodes $\mathcal{M}_i^2 (i \in \mathcal{I}(u))$ (denoted as a dotted square). The result of second-order maximum matching of the in-neighbour matrix of node pair (u, v) is as follows:*

$$\mathcal{M}^2(u,v) = \sum_{i\in\mathcal{I}(u)} \mathcal{M}_i^2 = \mathcal{M}_a^2 + \mathcal{M}_b^2 + \mathcal{M}_c^2$$

$\mathcal{M}_b^2$ and $\mathcal{M}_c^2$ can successfully extract information from nodes g and h (red area), which are ignored by the first-order maximum matching (solid square) adopted by RoleSim. Based on the values of top $\Gamma = 2$ maximum matching, we can generate the role similarity scores for node pair (u, v) using Eq. (5.4).

It is worth mentioning that the FaRS algorithm not only captures the information from the first-order maximum matching of the node pair in-neighbour matrix (green area) but also extracts information from the red area by generating the second-order maximum matching. In Example 17, the Γ has been set as 2, and it already captures more useful information (red area) than RoleSim. Actually, the parameter Γ in the FaRS algorithm ranges to $[1, n]$, and the larger the parameter Γ is, the more information will be extracted from node pair in-neighbour matrix and the more accurate FaRS will be.

Lemma 4. *Given an in-neighbour matrix of a node pair $(u, v) \in G$, $\mathcal{M}^1$ is the top bound of $\mathcal{M}^{\Gamma}$ $(1 \leq \Gamma \leq mindeg^-(u, v))$, and they are in decreasing order, i.e., The order of top-k perfect matching is as follows: $\mathcal{M}^1 \geq \mathcal{M}^2 \geq \cdots \geq \mathcal{M}^{\gamma}$.*

Proof. The particular reason for Lemma 4 is the requirement of the maximum matching technique. According to the definition of maximum matching in Definition 7, the maximum matching technique is the method to find the perfect matching of a matrix; thus, $\mathcal{M}^1$ is the best matching, and it is exact and unique. For the second-order maximum matching of a matrix, the matched values are different from the first-order maximum matching. The second-order maximum matching, which is based on the first maximum matching, changes several node pairs from the first maximum matching node-pairs; it re-matches them to find a new best matching (apart from the matching found in $\mathcal{M}^1$). Thus, $\mathcal{M}^2 \leq \mathcal{M}^1$. The circumstance is the same as the rest Γ^{th} $(1 < \Gamma < mindeg^-(u,v))$ order maximum matching, so we have $\mathcal{M}^1 \geq \mathcal{M}^2 \geq \cdots \geq \mathcal{M}^{\gamma}$. $\square$

The coefficients of top Γ^{th} order maximum matching in Equation 5.4 are $(1, \lambda, \ldots, \lambda^{\Gamma-1})$, respectively, which are in decreasing order as well.

Computation Of FaRS.

The value of the FaRS can be calculated iteratively and satisfies convergence. Here, we define the total iteration numbers of the algorithm is K, and the each iteration is represented by k $(K = maximum(k))$. The computation phases are presented as follows.

The first phase is the initialisation. Set $\mathbf{RS}_0^{\Gamma} = \mathbf{ones}_{n\times n}$. Then the single-source FaRS role similarity scores with respect to query q at iteration k are computed by the following:

$$\begin{aligned} \mathbf{RS}_k^{\Gamma}(:,q) = (1-C)(\mathcal{M}_{k-1}^1(:,q) + \lambda \cdot \mathcal{M}_{k-1}^2(:,q) + \\ \cdots + \lambda^{(\Gamma-1)} \cdot \mathcal{M}_{k-1}^{\Gamma}(:,q)) \oslash (1 + \lambda + \cdots + \lambda^{(\Gamma-1)})\mathbf{maxdeg}^-(:,q) + C \end{aligned} \tag{5.5}$$

The second phase is repeated until the result is convergent.

Note that the algorithm computes the role similarity scores by using the top Γ maximum matching values. The other method of capturing more information from the remain matrix is to remove the top best matching values from the in-neighbours matrix first and then to find the maximum matching of the remaining matrix (we named the role similarity algorithm using this method as FaRS_N). According to the experimental results conducted in Section 5.6, the accuracy of FaRS_N is not as good as FaRS. Therefore, our algorithm chooses the top Γ maximum matching method to retrieve the role similarity scores with respect to a query. As mentioned before, the parameter Γ is range to $[1, n]$. The different values of the parameter Γ do not affect the properties of the FaRS algorithm. Thus, we next details the FaRS algorithm with $\Gamma = 2$, and we called it the Sec-RoleSim algorithm. Furthermore, we evaluat the axiomatic properties of the FaRS algorithm by using Sec-RoleSim as an example to save space.

5.3.2 Sec-RoleSim: Role-Based Similarity Search algorithm Based On Graph Topology

In this subsection, we have illustrated the Sec-RoleSim algorithm in details first. Then, by using Sec-RoleSim as the example, we introduced some axiomatic properties of FaRS and proved them.

The definition of Sec-RoleSim is presented here first.

Theorem 16. *Given a graph $G = (\mathbf{V}, \mathbf{E})$ and a query q, the role similarity scores with respect to q can be generated as follows:*

$$\begin{aligned}\mathbf{RS}^2(:,q) = (1-C)\max_{\mathcal{M}^1(:,q)} \sum\nolimits_{(x,y)\in\mathcal{M}^1(:,q)} RS^2(x,y) + \\ \lambda \cdot \max_{\mathcal{M}^2(:,q)} \sum\nolimits_{(x,y)\in\mathcal{M}^2(:,q)} RS^2(x,y) \\ \oslash(1+\lambda)(\mathbf{deg}^-_{i=1:n} + \mathbf{deg}^-_q - |\mathbf{M}(:,q)|) + C\end{aligned} \tag{5.6}$$

Next, we showed the computational phases of Sec-RoleSim based on graph topology as follows. First, the initial value of Sec-RoleSim algorithm was set as $\mathbf{RS}^2_0 = \mathbf{ones}_{n\times n}$, where n is the number of nodes in the graph.

$$\begin{aligned}\mathbf{RS}^2_k(:,q) = (1-C)(\mathcal{M}^1_{k-1}(:,q) + \lambda \cdot \mathcal{M}^2_{k-1}(:,q)) \\ \oslash(1+\lambda)\mathbf{maxdeg}^-(:,q) + C\end{aligned} \tag{5.7}$$

After presenting the FaRS algorithm, its convergence is discussed next.

Theorem 17. ***Convergence:*** *Given a directed graph G and any query $q \in G$, the Sec-RoleSim role similarity search algorithm is converged with the initialisation of $RS^2_{k=0} = \mathbf{Ones}_{n\times n}$, and the iterative computation of the Sec-RoleSim algorithm with respect to query q at iteration k satisfies the following:*

$$\lim_{k\to\infty} \mathbf{RS}^2_k(:,q) = \mathbf{RS}^2(:,q)$$

Proof. Here, we need to illustrate that as k approaches infinity, the role similarity scores of $\mathbf{RS}^2_k(:,q)$ converge to $\mathbf{RS}^2(:,q)$. First, we prov the convergence of node-pair role similarity search score, which means if a node i from $\mathbf{V}$ is randomly selected, $\lim_{k\to\infty} RS^2_k(i,q) = RS^2(i,q)(i \in \mathbf{V})$.

It can be observed that

$$\begin{aligned}&\mathcal{M}_k(i,q) - \mathcal{M}(i,q) \\ &= \sum\nolimits_{(x,y)\in\mathcal{M}_{k-1}(i,q)} RS^2_{k-1}(x,y) - \sum\nolimits_{(x',y')\in\mathcal{M}(i,q)} RS^2(x',y')\end{aligned} \tag{5.8}$$

Based on Lemma 4, $\mathcal{M}(i,q)$ is larger than $\mathcal{M}_{k-1}(i,q)$, so Eqs. (5.8) can be written as follows:

$$\leq mindeg^{-}(i,q)\max\nolimits_{(x,y)\in M_{k-1}(i,q)}(RS^2_{k-1}(x,y) - RS^2(x,y)) \tag{5.9}$$

Then, the difference between $RS^2_k(i,q)$ and $RS^2(i,q)$ can be expressed as follows: (Eq (5.10)). Here, $RS^2_k(i,q)$ denotes the Sec-RoleSim role similarity scores with respect to query q at k^{th} iteration:

$$\begin{aligned}\varepsilon^k(i,q) &= RS^2_k(i,q) - RS^2(i,q) \\ &= C\frac{\mathcal{M}^1_{k-1}(i,q) + \lambda\mathcal{M}^2_{k-1}(i,q)}{(1+\lambda)maxdeg^{-}(i,q)} + (1-C) \\ &\quad - (C\frac{\mathcal{M}^1(i,q) + \lambda\mathcal{M}^2(i,q)}{(1+\lambda)maxdeg^{-}(i,q)} + (1-C)) \\ &= C\frac{(\mathcal{M}^1_{k-1}(i,q) - \mathcal{M}^1(i,q)) + \lambda(\mathcal{M}^2_{k-1}(i,q) - \mathcal{M}^2(i,q))}{(1+\lambda)maxdeg^{-}(i,q)}\end{aligned} \tag{5.10}$$

Substitute Eqs. (5.9) into Eqs. 5.10:

$$\begin{aligned}&\leq C(\frac{mindeg^{-}(i,q)\max_{(x,y)\in M^1_{k-1}(i,q)}(RS^2_{k-1}(x,y) - RS^2(x,y))}{(1+\lambda)maxdeg^{-}(i,q)} \\ &\quad + \lambda\frac{mindeg^{-}(i,q)\max_{(x,y)\in M^2_{k-1}(i,q)}(RS^2_{k-1}(x,y) - RS^2(x,y))}{(1+\lambda)maxdeg^{-}(i,q)}) \\ &\leq C\frac{(1+\lambda)mindeg^{-}(i,q)\max_{(x,y)\in M^1_{k-1}(i,q)}(RS^2_{k-1}(x,y) - RS^2(x,y))}{(1+\lambda)maxdeg^{-}(i,q)}\end{aligned} \tag{5.11}$$

assume that:

$$\begin{aligned}&RS^2_{k-1}(x^*,y^*) - RS^2(x^*,y^*) \\ &= \max\nolimits_{(x,y)\in M^1_{k-1}(i,q)}(RS^2_{k-1}(x,y) - RS^2(x,y))\end{aligned} \tag{5.12}$$

Thus, combining Eqs. (5.10) and Eqs. (5.12), we obtain:

$$\begin{aligned}
\varepsilon_k(i,q) &= RS_k^2(i,q) - RS^2(i,q) \\
&\leq C\frac{mindeg^-(i,q)}{maxdeg^-(i,q)}(RS_{k-1}^2(x_{k-1}^*,y_{k-1}^*) - RS^2(x_{k-1}^*,y_{k-1}^*)) \\
&\because \frac{mindeg^-(i,q)}{maxdeg^-(i,q)} \leq 1 \\
&\leq C(RS_{k-1}^2(x_{k-1}^*,y_{k-1}^*) - RS^2(x_{k-1}^*,y_{k-1}^*)) \qquad (5.13) \\
&\quad \cdots \\
&\leq C_k(RS_0^2(x_0^*,y_0^*) - RS^2(x_0^*,y_0^*)) \\
&\leq C_{k+1} \quad \because RS^2(x_0^*,y_0^*) \geq C
\end{aligned}$$

Since for any random node $i(i \in \mathbf{V})$, $\lim_{k\to\infty} RS_k^2(i,q) = RS^2(i,q)(i \in \mathbf{V})$ is proved.
Thus, $\lim_{k\to\infty} \mathbf{RS}_k^2(:,q) = \mathbf{RS}^2(:,q)$ is true. □

As we mentioned at the beginning of Subsection 5.3.2, the structural and properties of FaRS and Sec-RoleSim are the same. Sec-RoleSim is the specical case of FaRS with $\Gamma = 2$. Thus, the convergence proof of Sec-RoleSim can be easily extended to FaRS with different values of Γ. The proof of FaRS convergence has been omitted here to save space. In conclusion, FaRS is also a convergence algorithm.

5.3.3 The Axiomatic Properties of Sec-RoleSim

Recall that the RoleSim algorithm (Rothe & Schütze, 2014) has several axiomatic properties, including symmetry, monotone convergence, boundedness and triangle inequality. FaRS and Sec-RoleSim also have the axiomatic properties that are similar to that of the RoleSim algorithm. Next, we detailed these axiomatic properties of FaRS (Sec-RoleSim) along with proof. Since FaRS and Sec-RoleSim are essentially the same and the difference between them is the value of Γ, the proof of each property is generated using Sec-RoleSim for easier readability.

Theorem 18. ***Symmetry:*** *Given a graph $G = (\mathbf{V}, \mathbf{E})$ and a randomly selected query q, the role similarity scores generated by Eqs. 5.7 satisfy the following:*

$$\mathbf{RS}_k^2(:,q) = \mathbf{RS}_k^2(q,:)$$

Proof. We prove Theorem 18 by using the computation formula of Sec-RoleSim. In Eqs. (5.7), the role similarity scores are generated by extracting the information from the node-pair in-neighbour matrix. For any node $u \in \mathbf{V}$, the in-neighbour matrix of node-pair (u, q) and node-pair (q, u) are the same. Thus, the top two maximum matching results of node-pair (u, q) and node-pair (q, u) are the same, i.e., $\mathcal{M}^1(u, q) = \mathcal{M}^1(q, u)$ and $\mathcal{M}^2(u, q) = \mathcal{M}^2(q, u)$. Substituting the idea into Eqs. (5.7), we have the following:

$$\begin{aligned} RS_k^2(u, q) &= (1 - C)\frac{\mathcal{M}_{k-1}^1(u, q) + \lambda\mathcal{M}_{k-1}^2(u, q)}{(1 + \lambda)maxdeg^-(u, q)} \\ &= (1 - C)\frac{\mathcal{M}_{k-1}^1(q, u) + \lambda\mathcal{M}_{k-1}^2(q, u)}{(1 + \lambda)maxdeg^-(q, u)} \\ &= RS_k^2(q, u) \end{aligned}$$

Since node u belongs to $\mathbf{V}$, $\mathbf{RS}_k^2(:, q) = \mathbf{RS}_k^2(q, :)$. □

Theorem 19. ***Monotone Convergence:*** *Given a graph $G = (\mathbf{V}, \mathbf{E})$, a randomly selected query q and an iteration number k, the role similarity scores generated by Eqs. 5.7 satisfy the following:*

$$\mathbf{RS}_k^2(:, q) \leq \mathbf{RS}_{k-1}^2(:, q)$$

Proof. We proved Theorem 19 by using the mathematical induction method. We randomly selected a node $u (u \in \mathbf{V})$ and aimed to prove $\mathbf{RS}_k^2(u, q) \leq \mathbf{RS}_{k-1}^2(u, q)$. First, the role similarity values are initialised to be 1 when $k = 0$. Therefore, we had $\mathbf{RS}_0^2(u, q) = 1$. For the next iteration $k = 1$, Eqs. (5.7) for the node-pair (u, q) at iteration $(k = 1)$ was as follows:

$$\begin{aligned} RS_1^2(u, q) &= (1 - C)\frac{\mathcal{M}_0^1(u, q) + \lambda\mathcal{M}_0^2(u, q)}{(1 + \lambda)maxdeg^-(u, q)} + C \\ &\leq (1 - C)\underbrace{\frac{(1 + \lambda)mindeg^-(u, q)}{(1 + \lambda)maxdeg^-(u, q)}}_{\leq 1} + C \\ &\leq 1 = RS_0^2(u, q) \end{aligned}$$

Thus, Theorem 19 holds when $k = 1$.

Next, we assumed that Theorem 19 is true at the iteration k, i.e., $\mathbf{RS}_k^2(u, q) \leq \mathbf{RS}_{k-1}^2(u, q)$. We then set the iteration number to be $(k + 1)$. Eqs. (5.7) for the node-pair (u, q) at iteration $(k + 1)$ is as follows:

$$RS_{k+1}^2(u, q) = (1 - C)\frac{\mathcal{M}_k^1(u, q) + \lambda\mathcal{M}_k^2(u, q)}{(1 + \lambda)maxdeg^-(u, q)} + C \tag{5.14}$$

$\mathcal{M}_k^1(u,q)$ and $\mathcal{M}_k^2(u,q)$ are generated from the values of RS_{k-1}^2. Since $\mathbf{RS}_k^2(:,q) \leq \mathbf{RS}_{k-1}^2(:,q)$, so Eqs. 5.14 can be written as follows:

$$\begin{aligned} RS_{k+1}^2(u,q) &\leq (1-C)\frac{\mathcal{M}_{k-1}^1(u,q)+\lambda\mathcal{M}_{k-1}^2(u,q)}{(1+\lambda)maxdeg^-(u,q)}+C \\ &= RS_k^2(u,q) \end{aligned}$$

We have $RS_{k+1}^2(u,q) \leq RS_k^2(u,q)$. Since $u \in \mathbf{V}$, $\mathbf{RS}_k^2(:,q) \leq \mathbf{RS}_{k-1}^2(:,q)$.

Therefore, we have proved that if Theorem 19 is true at the iteration K, the next iteration $(k+1)$ can also be generated. Thus, Theorem 19 is true for all K. □

Theorem 20. ***Boundedness:*** *Given a graph $G = (\mathbf{V}, \mathbf{E})$ and a randomly selected query q, the role similarity scores generated by Eqs. 5.7 satisfy the following:*

$$\mathbf{C}_{n\times 1} \leq \mathbf{RS}_k^2(:,q) \leq \mathbf{1}_{n\times 1}$$

$\mathbf{C}_{n\times 1}$ is a vector whose values are all C, and $\mathbf{1}_{n\times 1} = ones(n,1)$.

Proof. We proved Theorem 20 by showing that for any node $u \in \mathbf{V}$, $C \leq \mathbf{RS}_k^2(u,q) \leq 1$. First, we initialised $\mathbf{RS}_k^2 = ones(n,n)$, and n is the number of nodes of the graph. Eqs. 5.7 is as follows:

$$\begin{aligned} RS_k^2(u,q) &= (1-C)\frac{\mathcal{M}_{k-1}^1(u,q)+\lambda\mathcal{M}_{k-1}^2(u,q)}{(1+\lambda)maxdeg^-(u,q)}+C \\ &\leq (1-C)\underbrace{\frac{(1+\lambda)mindeg^-(u,q)}{(1+\lambda)maxdeg^-(u,q)}}_{0\leq * \leq 1}+C \\ &\leq 1 \end{aligned}$$

And if the in-degree of node pair (u,q) equals zero, then $RS_k^2(u,q) = C$. Thus, $C \leq RS_k^2(u,q) \leq 1$ is true. Since node $u \in \mathbf{V}$, $\mathbf{C}_{n\times 1} \leq \mathbf{RS}_k^2(:,q) \leq \mathbf{1}_{n\times 1}$. □

Theorem 21. ***Triangle inequality:*** *Given a graph $G = (\mathbf{V}, \mathbf{E})$ and a randomly selected query q, for any nodes $(a,b) \in \mathbf{V}$, the role similarity scores satisfy the following:*

$$d_k(a,b) \leq d_k(a,q) + d_k(b,q) \qquad (5.15)$$

where $d_k(a,q) = 1 - RS_k^2(a,q)$.

Proof. We prove Theorem 21 by using the mathematical induction method. Since $d_k(a,q) = 1 - RS_k^2(a,q)$, Eqs. (5.15) can be rewritten as follows:

$$\begin{aligned}
&d_k(a,q) + d_k(b,q) \leq d_k(a,b) \\
&\qquad\Updownarrow \quad d_k(a,q) = 1 - RS_k^2(a,q) \\
&1 - RS_k^2(a,q) + 1 - RS_k^2(b,q) - 1 + RS_k^2(a,b) \leq 0 \\
&\qquad\Updownarrow \\
&RS_k^2(a,q) + RS_k^2(b,q) - RS_k^2(a,b) \leq 1
\end{aligned} \tag{5.16}$$

Then, we need Eqs. 5.16 to hold.

To do that, first we initialise the role similarity scores at the iteration $k = 0$, as $RS_0^2 = \mathbf{Ones}_{n\times n}$. Here, n is the number of nodes in the graph. When $k = 0$, Eqs. 5.16 can be written as follows:

$$RS_0^2(a,q) + RS_0^2(b,q) - RS_0^2(a,b) = 1 + 1 - 1 \leq 1$$

Thus, when iteration $k = 0$, Eqs. 5.16 holds. Next, we assume that Eqs. 5.16 holds at the k^{th} iteration; then, we need to prove that Eqs. 5.16 also holds at the $(k+1)^{th}$ iteration.

$$\begin{aligned}
&RS_{k+1}^2(a,q) + RS_{k+1}^2(b,q) - RS_{k+1}^2(a,b) \\
&= (1-C)\frac{\mathcal{M}_k^1(a,q) + \lambda\mathcal{M}_k^2(a,q)}{(1+\lambda)maxdeg^-(a,q)} + C + (1-C)\frac{\mathcal{M}_k^1(b,q) + \lambda\mathcal{M}_k^2(b,q)}{(1+\lambda)maxdeg^-(b,q)} + C \\
&\quad - (1-C)\frac{\mathcal{M}_k^1(a,b) + \lambda\mathcal{M}_k^2(a,b)}{(1+\lambda)maxdeg^-(a,b)} - C \\
&= \frac{(1-C)}{(1+\lambda)}\left(\begin{array}{l}\dfrac{\mathcal{M}_k^1(a,q) + \lambda\mathcal{M}_k^2(a,q)}{maxdeg^-(a,q)} + \dfrac{\mathcal{M}_k^1(b,q) + \lambda\mathcal{M}_k^2(b,q)}{maxdeg^-(b,q)} \\ - \dfrac{\mathcal{M}_k^1(a,b) + \lambda\mathcal{M}_k^2(a,b)}{maxdeg^-(a,b)}\end{array}\right) + C \\
&\leq \frac{(1-C)}{(1+\lambda)}\left(\frac{(1+\lambda)mindeg^-(a,q)}{maxdeg^-(a,q)} + \frac{(1+\lambda)mindeg^-(b,q)}{maxdeg^-(b,q)} - \frac{(1+\lambda)mindeg^-(a,b)}{maxdeg^-(a,b)}\right) \\
&\qquad = (1+C)\left(\frac{mindeg^-(a,q)}{maxdeg^-(a,q)} + \frac{mindeg^-(b,q)}{maxdeg^-(b,q)} - \frac{mindeg^-(a,b)}{maxdeg^-(a,b)}\right) + C
\end{aligned} \tag{5.17}$$

There are three cases to be considered for Eqs.5.17, including the following: $deg_a^- \leq deg_b^- \leq deg_q^-$, $deg_b^- \leq deg_a^- \leq deg_q^-$ and $deg_a^- \leq deg_q^- \leq deg_b^-$. For the case $deg_a^- \leq deg_b^- \leq deg_q^-$, Eqs.5.17 can be written as follows:

$$\leq (1-C)\left(\frac{mindeg^-(a,q)}{maxdeg^-(a,q)}+\frac{mindeg^-(b,q)}{maxdeg^-(b,q)}-\frac{mindeg^-(a,b)}{maxdeg^-(a,b)}\right)+C$$

$$\Updownarrow \deg_a^- \leq \deg_b^- \leq \deg_q^-$$

$$=(1-C)\left(\frac{\deg_a^-}{\deg_q^-}+\frac{\deg_b^-}{\deg_q^-}-\frac{\deg_a^-}{\deg_b^-}\right)+C$$

$$=(1-C)\left(\frac{\deg_a^-+\deg_b^-}{\deg_q^-}-\frac{\deg_a^-}{\deg_b^-}\right)+C$$

$$\Updownarrow \deg_a^- \leq \deg_b^- \leq \deg_q^-$$

$$\leq (1-C)\left(\frac{\deg_a^-+\deg_b^-}{\deg_b^-}-\frac{\deg_a^-}{\deg_b^-}\right)+C \quad =(1-C)\left(\frac{\deg_b^-}{\deg_b^-}\right)+C \quad =1$$

Thus, $RS^2_{k+1}(a,q)+RS^2_{k+1}(b,q)-RS^2_{k+1}(a,b)\leq 1$ holds when $deg_a^- \leq deg_b^- \leq deg_q^-$.

Next, we consider the second case where $deg_b^- \leq deg_a^- \leq deg_q^-$, and Eqs.5.17 can be written as follows:

$$=(1-C)\left(\frac{\deg_a^-}{\deg_q^-}+\frac{\deg_b^-}{\deg_q^-}-\frac{\deg_b^-}{\deg_a^-}\right)+C$$

$$=(1-C)\left(\frac{\deg_a^-+\deg_b^-}{\deg_q^-}-\frac{\deg_b^-}{\deg_a^-}\right)+C$$

$$\Updownarrow \deg_b^- \leq \deg_a^- \leq \deg_q^-$$

$$\leq (1-C)\left(\frac{\deg_a^-+\deg_b^-}{\deg_a^-}-\frac{\deg_b^-}{\deg_a^-}\right)+C \quad =(1-C)\left(\frac{\deg_a^-}{\deg_a^-}\right)+C \quad =1$$

Thus, $RS^2_{k+1}(a,q)+RS^2_{k+1}(b,q)-RS^2_{k+1}(a,b)\leq 1$ also holds when $deg_b^- \leq deg_a^- \leq deg_q^-$.

The last case was $deg_a^- \leq deg_q^- \leq deg_b^-$, and Eqs.5.17 can be written as follows:

$$=(1-C)\left(\frac{\deg_a^-}{\deg_q^-}+\frac{\deg_q^-}{\deg_b^-}-\frac{\deg_a^-}{\deg_b^-}\right)+C$$

$$=(1-C)\left(\frac{\deg_q^- - \deg_a^-}{\deg_b^-}+\frac{\deg_a^-}{\deg_q^-}\right)+C$$

$$\Updownarrow \deg_a^- \leq \deg_q^- \leq \deg_b^-$$

$$\leq (1-C)\left(\frac{\deg_q^- - \deg_a^-}{\deg_q^-}+\frac{\deg_a^-}{\deg_q^-}\right)+C \quad =(1-C)\left(\frac{\deg_q^-}{\deg_q^-}\right)+C \quad =1$$

Thus, $RS^2_{k+1}(a,q)+RS^2_{k+1}(b,q)-RS^2_{k+1}(a,b)\leq 1$ holds when $deg_a^- \leq deg_q^- \leq deg_b^-$.

The proof of three cases above showed that $RS^2_{k+1}(a,q)+RS^2_{k+1}(b,q)-RS^2_{k+1}(a,b)\leq 1$, which means $RS^2_k(a,q)+RS^2_k(b,q)-RS^2_k(a,b)\leq 1$. This implies $d^k(a,q)+d^k(b,q)\leq d^k(a,b)$. □

In this section, the FaRS and Sec-RoleSim algorithm are defined. Theorem 17 guarantees the uniqueness and exactness of Sec-RoleSim, Theorem 18 to Theorem 21 present some properties of FaRS and Sec-RoleSim. In the next section, we introduce two speed-up techniques for accelerating the Sec-RoleSim algorithm based on graph topology.

5.4 EFFICIENT COMPUTATION

In this section, we introduce two speed-up techniques for accelerating the computation of FaRS. These proposed speed-up approaches can considerably minimise the number of calls of the maximum matching algorithm in FaRS, and also extract "shared" information to reduce repetitive operations. The final accelerate algorithm is named as Opt_FaRS.

5.4.1 Pruning Approach

Opt_FaRS consists of two stages: the pre-processing measure, and the iterative computation. The pre-processing measure includes the tracking path extraction and the computation of the `candidate pool`.

Definition 8 (**Multi-Hop Backward Tracking Path**). *Given a connected graph $G = (\mathbf{V}, \mathbf{E})$, a query $q \in \mathbf{V}$ and the number of iterations K defined in FaRS, the tracking path P with respect to query q is $\boldsymbol{P(q)} =< \boldsymbol{p}^1, \boldsymbol{p}^2 \ldots, \boldsymbol{p}^L >$, where p^i is the i^{th}-hop backward tracking nodes set with respect to query q, and it is iteratively defined as follows:*

$$\begin{cases} \boldsymbol{p}^1 = \{q\} \\ \boldsymbol{p}^l = \{\mathcal{I}(x^1) \bigcup \mathcal{I}(x^2) \cdots \bigcup \mathcal{I}(x^{|\boldsymbol{p}^{l-1}|}) \mid x^1, x^2, \ldots x^{|\boldsymbol{p}^{l-1}|} \in \boldsymbol{p}^{l-1}\} \end{cases} \tag{5.18}$$

L is the actual iteration numbers of the FaRS algorithm before the convergence, which can also be called the level of the tracking path. It satisfies the condition of $1 \leq L \leq K$. The repeated nodes in $\boldsymbol{p}^l$ need to be removed from the set to retain the uniqueness.

Example 18. *Given a graph G with five nodes, a query $q = d$, and the number of iteration $K = 5$ in Figure 5.4 (the left side), the generated track paths according to Eqs.*(5.18) *are presented in Figure 5.4 (the right side).*

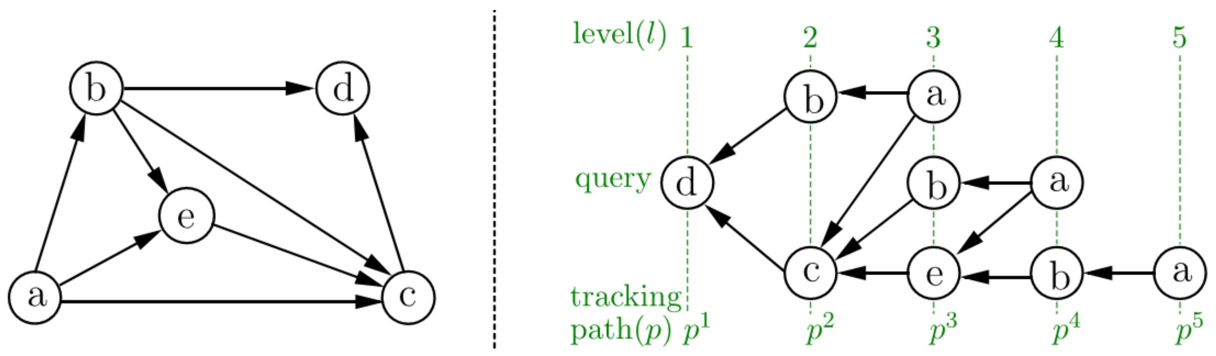


Figure 5.4: Left side: Example of Graph G. Right side: Multi-Hop Backward Tracking Path of the Graph G

In Figure 5.4, the track path is a traverse staring from the query node d. Based on Definition 8, the second track path is the in-neighbour nodes of query node d; thus, we have $\boldsymbol{p}^2 = \{b, c\}$. *The tracking path is determined by the query node and the structure of the graph. To generate* $\boldsymbol{p}^3$*, according to Eqs.*(5.18)*, we have the following:* $\boldsymbol{p}^3 = \mathcal{I}(\boldsymbol{p}^2) = \mathcal{I}(b) \cup \mathcal{I}(c) = \{a, a, b, c\}$. *Removing the repeated elements, the final result is* $\boldsymbol{p}^3 = \{a, b, c\}$. $\boldsymbol{p}^4$ *and* $\boldsymbol{p}^5$ *can be calculated similarly. It is worth mentioning that the tracking path stopped at* $\boldsymbol{p}^5 = [a]$ *and the in-degree of node a is zero, which guarantee the convergence.*

The purposes of extracting the tracking path of G with respect to query q are twofold: firstly, it can significantly reduce the calculation of some useless information. After getting the tracking path, in each iteration of role similarity scores calculation, FaRS does not need to calculate the role similarity value of all node pairs any more. Instead, it only needs to calculate the role similarity scores of the corresponding columns, and the corresponding columns are determined by the nodes set $\mathbf{p}^l$. secondly it is possible to reduce the number of iterations of FaRS. When the level number L of the tracking path is less than the given iteration number K, we only need to iterate L times. This is because that according to the graph structure, after iterating L times, the role similarity scores converge, which means $Fa\_\mathbf{RS}_k^\Gamma = Fa\_\mathbf{RS}_L^\Gamma$ $(L \leq k \leq K)$.

Leveraging the track path of graph G, we illustrate how to generate candidate pools (**CP**) by using the tracking path elements as the index.

The basic idea is that a candidate pool is generated for each iteration with respect to each tracking path. Each candidate pool is a role similarity search scores matrix that consists of all the necessary information for the next iteration of FaRS calculation. **CP** is generated by iterations, and the number of iterations is determined by the length of the tracking path L and iteration number K. The elements of each tracking path, $\mathbf{p}^l$, determines the column of the corresponding **CP**.

In addition, according to the definition of the FaRS algorithm, it can be seen that FaRS only captures the information of the node pair in-neighbour matrix for the calculation of the role similarity score of the node pair. Therefore, the similarity scores of these nodes without out-neighbours will not affect the results of other nodes. Therefore, ignoring the calculation of these nodes will not affect the final single-source role similarity scores. Based on this observation, the proposed Opt_FaRS algorithm ignores these node rows from each candidate pool, which greatly reduces the computational complexity. More details of the Opt_FaRS algorithm have been discussed later in this section.

The candidate pool can be formally defined as follows:

Definition 9 (**Candidate pool**)**.** *Given a track path* $\boldsymbol{P} =< \boldsymbol{p}^1, \boldsymbol{p}^2 \ldots, \boldsymbol{p}^L >$ *of a connected graph* G*, one candidate pool is defined for each* $\mathbf{p}^l$ *in* $\boldsymbol{P}$*. All candidate pools have a fixed size of rows, which consist of all the nodes in* G *with out-neighbours, denoted as* $\mathbf{out}$ $(\mathbf{out} = \{x \in \mathbf{V} | \mathcal{O}(x) \neq \emptyset\})$*. The column of each candidate pool is determined by the corresponding tracking path element* $\mathbf{p}^l$*. At the iteration* k *of FaRS, the candidate pool,* $\mathbf{CP}_k$*, can be repre sented as*

$$\mathbf{RS}_k^{\Gamma}(\mathbf{out}, \boldsymbol{p}^l) = \mathbf{CP}_k^{\Gamma} \qquad (l = K - k + 1, \quad 1 < k \leq L)$$

$\mathbf{RS}_k^{\Gamma}$ *is the role similarity score matrix of all pairs of the graph generated by the FaRS algorithm.* $\mathbf{CP}_k^{\Gamma}$ *is the candidate pool with respect to iteration* k*.*

The size of $\mathbf{CP}_k^{\Gamma}$ generally is much smaller than $\mathbf{RS}_k^{\Gamma}$. This is because the size of the nodes with out-neighbours (candidate pool's row) is smaller or equal to the total number of nodes in a graph, and the length of each element of a track path (candidate pool's column) are much smaller then the total number of nodes in a graph (i.e., $|\mathbf{out}| \leq n$ and $|\mathbf{p}^l| \ll n$ where n is the number of nodes in the graph). The previous research on RoleSim need to calculate all the node pairs' $(n \times n)$ role similarity scores at each iteration. By introducing the concept of candidate pools, for each iteration, the computation cost of our Opt_FaRS algorithm is reduced by the range $(|\mathbf{out}| \times |\mathbf{p}^l|)$ of information retrieval.

Next, based on the computation formula of FaRS (Eqs. 5.5) and the definition of the candidate pool, Opt_FaRS, an efficient single-source similarity search algorithm is proposed. Mathematically, Opt_FaRS is shown as follows:

Theorem 22. *Given a connected graph $G = (\mathbf{V}, \mathbf{E})$, a random query q and the track path $\boldsymbol{P} =< \boldsymbol{p}^1, \boldsymbol{p}^2 \ldots, \boldsymbol{p}^L >$ corresponding to q, the candidate pool with respect to the tracking path at iteration k can be updated with the following:*

$$\begin{aligned}&\mathbf{RS}_k^{\Gamma}(\mathbf{out}, j) = \mathbf{CP}_k^{\Gamma}(:, j) = (1 - C)(\mathcal{M}_{k-1}^1(:, j) + \lambda \mathcal{M}_{k-1}^2(:, j) + \\ &\cdots + \lambda^{\Gamma-1}\mathcal{M}_{k-1}^{\Gamma}(:, j)) \oslash (1 + \lambda + \cdots + \lambda^{\Gamma-1})\mathbf{maxdeg}^{-}(:, j) + C \qquad j \in \boldsymbol{p}^{L-k+1}\end{aligned} \tag{5.19}$$

j is a node in track path $\boldsymbol{p}^{L-k+1}$. $\boldsymbol{p}^{L-k+1}$ determine the column index of the candidate pool, so the candidate pool at iteration k is $\mathbf{RS}_k^{\Gamma}(\mathbf{out}, \boldsymbol{p}^{L-k+1}) = \mathbf{CP}_k^{\Gamma}$. $\mathbf{maxdeg}^{-}(:, j)$ is a vector whose values represent the maximum in-degree between node j and each node in $\mathbf{out}$, respectively. $\mathcal{M}_{k-1}^{\Gamma}(:, j)$ is a vector, the top Γ maximum weighted matching of the node pair (i, j) in-neighbour matrix, where $i \in \mathbf{out}$ at iteration $(k - 1)$.

The Opt_FaRS algorithm involves two steps. The first step is to retrieve the tracking path $\mathbf{P}$ of G, which starts from the query q. The next step is to generate the candidate pool with respect to the tracking path at iteration k. The iteration number k used in Eqs. (5.19) is also the order of the computation of Opt_FaRS. The order of algorithm computation is opposite to the order of tracking path $\mathbf{P}$. The number of total iterations K is equal to the track path level number L. Recalling the example in Figure 5.4, the algorithm should start from $\mathbf{p}^5(k = 1)$ to $\mathbf{p}^1(k = 5)$ and compute from $\mathbf{CP}_1^{\Gamma}(:, q)$ to $\mathbf{CP}_5^{\Gamma}(:, q)$. It is worth mentioning that the tracking path stopped at $\mathbf{p}^5 = [a]$, and the in-degree of node a is zero, which guarantees the convergence. Finally, the role similarity scores with respect to query q are as follows: $\text{Opt\_FaRS}(:, q) = \mathbf{RS}_K^{\Gamma}(:, q) = \mathbf{CP}_K^{\Gamma}$, here, the size of $\mathbf{CP}_{K-1}^{\Gamma}$ is $|\mathbf{out}| \times \mathbf{p}^2$, and the size of $\mathbf{CP}_K^{\Gamma}$ is $n \times 1(|\mathbf{out}| \leq n)$, then we set the value of difference index between n and $\mathbf{out}$ equal to C, This operation ensures that the $\mathbf{CP}_K^{\Gamma}$ and $\mathbf{RS}_K^{\Gamma}(:, q)$ sizes are consistent. Note that when $\mathbf{maxdeg}^{-}(\mathbf{out}, j) = 0$, the result of Eqs. 5.19 is equal to $\mathbf{C}_{\mathbf{out}\times 1}$.

Opt_FaRS omits the useless calculation of the FaRS algorithm without losing any accuracy. The tracking path schema can exactly extract the set of nodes that has an impact on the query's role similarity scores. The nodes set of the tracking path determines the column index of corresponding $\mathbf{CP}_k^{\Gamma}$ (j in Eqs. 5.19). Thus, Opt_FaRS only computes the relevant nodes' similarity scores of query q. Since the nodes without out-neighbours cannot influence other nodes' similarity scores, the calculation of these nodes in each iteration is omitted. The set of nodes with out-neighbours is represented as $\mathbf{out}$. The tracking path and $\mathbf{out}$ greatly reduce the computational complexity without losing accuracy. Since the initialism value of the Opt_FaRS algorithm is $\mathbf{ones}_{\mathbf{out}\times n}$, when $k=1$, the j^{th} column of the candidate pool can be computed by the following:

$$\mathbf{CP}_1^{\Gamma}(:,j) = (1-C)\mathbf{mindeg}^{-}(\mathbf{out},j) \oslash \mathbf{maxdeg}^{-}(\mathbf{out},j) + C \quad j \in L \tag{5.20}$$

We can learn from the first generation $\mathbf{CP}_1^{\Gamma}$ that the maximum matching algorithm is omitted. The maximum matching value of each bipartite graph is $\mathbf{mindeg}^{-}(|\mathbf{out}|,j)$, respectively. Since the elements of the initial Opt_FaRS is $\mathbf{1}_{\mathbf{out}\times n}$.

Example 19. *Recall Example 18 and Figure 5.4. We have the graph G, query $q=d$ and the tracking path P of the graph with respect to d. Given that the decay factor is $C=0.2$, $\lambda=0.5$, $\Gamma=2$ and the role similarity scores of $\mathbf{CP}_2^2$ at iteration $k=2$.*

$$\mathbf{CP}_2^2 = \begin{array}{c} \\ (a) \\ (b) \end{array} \begin{array}{cccc} (a) & (b) & (c) & (e) \\ \end{array}$$

$$\mathbf{CP}_2^2 = \begin{array}{c|cccc} & (a) & (b) & (c) & (e) \\ \hline (a) & 0.2 & 0.2 & 0.2 & 0.2 \\ (b) & 0.2 & 0.36 & 0.253 & 0.28 \end{array}^{T}$$

Based on this information, we demonstrate how to compute $\mathbf{CP}_3^2$ via Eqs.5.19 as follows.

First, we need find out the row and column index of $\mathbf{CP}_3^2$. According to the Figure 5.4, the nodes set of $\boldsymbol{p}^3 = [a,b,e]$; thus, the column index of $\mathbf{CP}_3^2$ is $[a,b,e]$. All the $\mathbf{CP}$ have the same row indexes as $\mathbf{out}$, which is unchanged from $\mathbf{CP}_2^2$ as $[a,b,c,e]$.

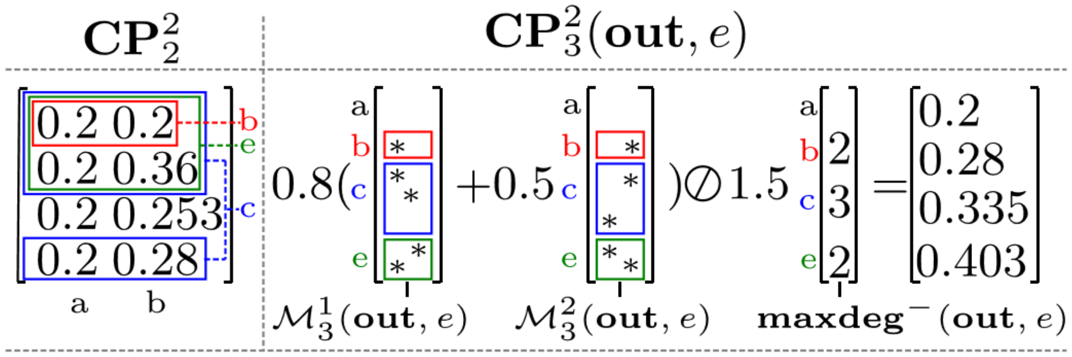
Next, we computed the role similarity scores of $\mathbf{CP}_3^2$ column by column.

1. ***(Column** a.**)*** *The in-degree of node a is zero, so we have $\mathbf{CP}_3^2(\mathbf{out},a) = \mathbf{C}_{4\times 1}$.*

2. ***(Column** b.**)*** *The second column of $\mathbf{CP}_3^2$ is b. The in-neighbour of node b is node a, and $deg_a^- = 0$; thus, $\mathbf{CP}_3^2(\mathbf{out},b)$ can be quickly calculated(Theorem 22), and the result is $[0.2, 0.36, 0.2533, 0.28]^T$.*

3. ***(Column** e.**)*** *Concerning the iterated computation of $\mathbf{CP}_3^2(\mathbf{out},e)$, details are as follows.*

   Here, the rectangles of different colours represent the in-neighbour matrix of different node pairs (red-(b,e), blue-(c,e),green-(e,e)). $\mathcal{M}_3^1(\mathbf{out},e)$ is the first-order maximum matching of each node pair at the iteration $k=3$.



$\mathbf{CP}_2^2$
$\mathbf{CP}_3^2(\mathbf{out}, e)$
0.2 0.2
0.2 0.36
0.2 0.253
0.2 0.28
a
b
c
e
$0.8($
$+0.5$
$)\oslash 1.5$
2
3
2
$=$
0.2
0.28
0.335
0.403
$\mathcal{M}_3^1(\mathbf{out}, e)$
$\mathcal{M}_3^2(\mathbf{out}, e)$
$\mathbf{maxdeg}^-(\mathbf{out}, e)$

Figure 5.5: Iterated Computation of $\mathbf{CP}_3^2(\mathbf{out}, e)$

Thus, it is composed of $[\mathcal{M}_3^1(a,e), \mathcal{M}_3^1(b,e), \mathcal{M}_3^1(c,e), \mathcal{M}_3^1(e,e)]^T$. *'$*$' is the maximum matching value of each in-neighbour matrix. Figure 5.5 presents the computation details of* $\mathbf{CP}_3^2(\mathbf{out}, e)$*, and the result is* $[0.2, 0.28, 0.335, 0.403]^T$.

In summary, the role similarity score of $\mathbf{CP}_3^2$ *has been retrieved as follows:*

$$\mathbf{CP}_3^2 = \begin{array}{c} \\ (a) \\ (b) \\ (e) \end{array} \begin{array}{c} \begin{array}{cccc} (a) & (b) & (c) & (e) \end{array} \\ \left[\begin{array}{cccc} 0.2 & 0.2 & 0.2 & 0.2 \\ 0.2 & 0.36 & 0.2533 & 0.28 \\ 0.2 & 0.28 & 0.335 & 0.2533 \end{array}\right]^T \end{array}$$

We can see from Figure 5.5 that the position of the first maximum matching value ($\mathcal{M}^1$) in each in-neighbour matrix is different from the second maximum matching value ($\mathcal{M}^2$). Thus, second maximum matching can capture more information from the node pair in-neighbour matrix. Furthermore, this example shows that the Opt_FaRS algorithm generate three column role similarity scores at iteration $k = 3$ rather than computing the role similarity scores of all node pairs, which significantly improve the efficiency of FaRS computation.

Next, we introduce some exceptional cases of Theorem 22, which also contribute positively to the speed-up.

It can be seen from Eqs.(5.19) that apart from $\mathbf{CP}_1^\Gamma$, the number of calls of the maximum matching algorithm for $\mathbf{CP}_k^\Gamma$ computation is large, which is time-consuming. In order to reduce the computational complexity of Opt_FaRS, two speed-up approaches for the candidate pool computation, one to optimise the column generation and one to optimise the row generation, have been presented next.

5.4.2 P-speedup approach

Based on Eqs.(5.19), the column indexes of candidate pools are determined by the corresponding track path. Therefore, we named our speed-up approach on column generation as the **P-speedup** approach.

There are two exceptional cases of the Opt_FaRS algorithm where Opt_FaRS can retrieve the role similarity scores without the need of using the maximum matching algorithm.

Definition 10 (exceptional cases). *Given a connected graph $G(\mathbf{V}, \mathbf{E})$,*

- ***One-hop*** *It includes all these nodes in G whose in-degrees are zero. We define a node i in G belonging to the one-hop set (denoted as $\boldsymbol{V}^{(1)}$), mathematically, as:*

$$\boldsymbol{V}^{(1)} = \{i \mid deg_i^- = 0, i \in V\}$$

- ***Two-hop*** *It includes all these nodes in G whose in-neighbours' in-degrees are all zero. We define the two-hop node set (denoted as $\boldsymbol{V}^{(2)}$), mathematically as:*

$$\boldsymbol{V}^{(1)} = \{i \mid deg_{\mathcal{I}(i)}^- = 0, i \in V\}$$

where the $\mathcal{I}(i)$ is the in-neighbour set of node i in the graph.

We observe that when a node of a graph belongs to the aforementioned exceptional cases, the one-hop set or two-hop set, the value of the candidate pool can be generated directly.

Lemma 5. *Given a connected graph $G(\mathbf{V}, \mathbf{E})$, query column $j (j \in \boldsymbol{p}^k)$,*

- *if the node j belongs to one-hop set $\mathbf{V}^{(1)}$, the candidate pool value $\mathbf{CP}_k^\Gamma(:, j)$ can be generated by*

$$\mathbf{CP}_k^\Gamma(:, j) = \mathbf{C}_{\mathbf{out} \times 1}$$

Note that the role similarity scores of $\mathbf{CP}_k^\Gamma(:, j)$ will remain the same at all iterations.

- *if the node j belongs to two-hop set $\mathbf{V}^{(2)}$, the candidate pool value $\mathbf{CP}_k^{\Gamma}(:,j)$ can be generated by*

$$\mathbf{CP}_k^{\Gamma}(:,j) = (1-C)(C\mathbf{mindeg}^{-}(\mathbf{out},j) \oslash \mathbf{maxdeg}^{-}(\mathbf{out},j) + \mathbf{1}_{\mathbf{out}\times 1})$$

  here, $\mathbf{1}_{\mathbf{out}\times 1}$ is a vector, whose size is $\mathbf{out} \times 1$. $\{\mathbf{out}\}^{th}$ entries of $\mathbf{1}_{\mathbf{out}\times 1}$ is 1, others are removed. In this exceptional case, the role similarity scores are convergent at the iteration $k = 2$.

5.4.3 Out-speedup approach

Based on Lemma 5, the computation of $\mathbf{CP}_k^{\Gamma}$ was optimised when the column index nodes belonged to the two exceptional cases. Next, we discuss that the computation of $\mathbf{CP}_k^{\Gamma}$ can be further improved based on certain rows of $\mathbf{CP}_k^{\Gamma}$ matrix. The row indexes of each candidate pool are determined by **out**, so the accelerate scheme on the row nodes is named as the **out-speedup** approach.

We observe that if a node needs to surf more than two hops to catch the root node of the graph, the similarity score needs to be generated by the maximum weighted matching algorithm. As the computation of maximum matching over large-scale graphs with high time complexity, we propose an optimisation strategy for speeding up the computation of maximum matching of the node pair in-neighbour matrix ($\mathcal{M}$) in Lemma 6.

Lemma 6. *Given a connected graph $G(\mathbf{V},\mathbf{E})$, a query column j ($j \in \mathbf{p}^k$), any node $i(i \in \mathbf{out})$ and an iteration number k,*

1. *if the node i belongs to one-hop set $\mathbf{V}^{(1)}$, the maximum weighted value $\mathcal{M}_k(i,j)$ can be generated by*

$$\mathcal{M}_k(i,j) = 0$$

2. *if the node i belongs to two-hop set $\mathbf{V}^{(2)}$, the maximum weighted value $\mathcal{M}_k(i,j)$ can be generated by*

$$\mathcal{M}_k(i,j) = C$$

3. *otherwise, the maximum weighted matching of $\mathbf{CP}_{k-1}$ is generated. Before introducing the computation method, we have introduced several notions. The maximum matching*

result of $\mathbf{CP}_{k-1}$ is denoted as $\mathcal{M}[\mathbf{CP}_{k-1}]$. The matched values set of maximum matching on $\mathbf{CP}_{k-1}$ is represented by $\widetilde{\mathcal{M}}[\mathbf{CP}_{k-1}]$. The in-neighbour matrix of node-pair (i,j) is defined as $\mathcal{B}_{ij}$. The maximum matching result of $\mathcal{B}_{ij}$ is $\mathcal{M}[\mathcal{B}_{ij}]$, and the matched values set of maximum matching on $\mathcal{B}_{ij}$ is $\widetilde{\mathcal{M}}[\mathcal{B}_{ij}]$. The matched values of $\mathbf{CP}_{k-1}$ in $\mathcal{B}_{ij}$ are defined as $\mathcal{M}[\mathcal{B}_{ij},\mathbf{CP}_{k-1}]$, the number of matched values of $\mathbf{CP}_{k-1}$ in $\mathcal{B}_{ij}$ is $|\mathcal{M}_k[\mathcal{B}_{ij},\mathbf{CP}_{k-1}]|$.

- *if the minimum value between deg_i^- and deg_j^- is equal to $|\mathcal{M}_k[\mathcal{B}_{ij},\mathbf{CP}_{k-1}]|$, we have*

$$\mathcal{M}_k(i,j) = \mathcal{M}_k[\mathbf{CP}_{k-1}]$$

- *if the minimum value between deg_i^- and deg_j^- is larger than $|\mathcal{M}_k[\mathcal{B}_{ij},\mathbf{CP}_{k-1}]|$,*
  - *the matched values in $|\mathcal{M}_k[\mathcal{B}_{ij},\mathbf{CP}_{k-1}]|$ are the maximum value of its column and row of the bipartite matrix $\mathcal{B}_{ij}$. The value of $\mathcal{M}_k(i,j)$ can be generated by*

$$\mathcal{M}_k(i,j) = sum(\mathcal{M}_k[\mathcal{B}_{ij},\mathbf{CP}_{k-1}]) + \mathcal{M}_k[\mathcal{B}_{ij}^{rem}]$$

    where $sum(\mathcal{M}_k[\mathcal{B}_{ij},\mathbf{CP}_{k-1}])$ is the sum of the matched values between the bipartite graph $\mathcal{B}_{ij}$ and $\mathcal{M}_k[\boldsymbol{CP}_{k-1}]$. We eliminate the matched elements' rows and columns from the bipartite graph. We define it as $\mathcal{B}_{ij}^{rem}$, and $\mathcal{M}_k(\mathcal{B}_{ij}^{rem})$ represent the maximum matching score of the remaining bipartite graph.
  - *otherwise, the value of $\mathcal{M}_k(i,j)$ is as follows:*

$$\mathcal{M}_k(i,j) = \mathcal{M}_k[\mathcal{B}_{ij}]$$

    where $\mathcal{M}_k[\mathcal{B}_{ij}]$ is the maximum matching value of the in-neighbour matrix $\mathcal{B}_{ij}$.

$\mathcal{M}_k(i,j)$ is the maximum matching value of node pair (i,j) in-neighbour matrix in the iteration k.

This speed-up approach can significantly improve the performance of the Opt_FaRS algorithm without sacrificing accuracy. For example, if a node pair (i,j) belongs to exceptional cases, the similarity score $\mathcal{M}_k(i,j)$ can be retrieved without the need of performing the maximum matching computation (Lemma 6. (1) (2)). Furthermore, when node pairs are not belong to these exceptional cases, we could generate $\mathcal{M}_{(i,j)}^k$ with minimum number of calls of the maximum matching computation. The call number is less than $|\mathbf{out}|$ rather than n times (n is the total number of nodes in graphs) (Lemma 6. (3)).

Algorithm 4: Opt_FaRS $(G, C, q, K, \lambda, \Gamma)$

```
Input  : a graph $G$, decay factor $C$, a query set $q$, #-iteration $K$, #-maximum matching $\Gamma$, $\lambda$
Output: exact role similarity scores of query $q$ at the iteration $K$.
2  generate the tracking path $\mathbf{P}$, and the actual iteration number $L = |\mathbf{P}|$;
4  $\mathbf{out} = \{x \in \mathbf{V} | \mathcal{O}(x) \neq \emptyset\}$;
6  Initialize $\mathbf{CP}_0^\Gamma = ones(\mathbf{out}, I(\mathbf{P}^L))$;
8  for $k = 1$ to $L$ do
10     foreach $j \in \boldsymbol{p}^{L-k+1}$ do
12         if $j \in \mathbf{V}^{(1)}$ then
13             $\mathbf{CP}_k(:, j) = [\mathbf{1} - \mathbf{C}]_{|\mathbf{out}| \times 1}$
15         else if $j \in \mathbf{V}^{(2)}$ then
16             $\mathbf{CP}_k^\Gamma(:, j) = (1 - C)(C\mathbf{mindeg}^-(\mathbf{out}, j) \oslash \mathbf{maxdeg}^-(\mathbf{out}, j) + \mathbf{1}_{\mathbf{out} \times 1})$
18         else
20             foreach $i \in \mathbf{out}$ do
22                 if $i \in \mathbf{V}^{(1)}$ then
24                     $\mathcal{M}_k(i, j) = 0$
26                 else if $i \in \mathbf{V}^{(2)}$ then
28                     $\mathcal{M}_k(i, j) = (1 - C)$
30                 else
31                     compute maximum matching of $\mathbf{CP}_{k-1}^\Gamma$
33                     if $mindeg^-(i, j) == |\mathcal{M}_k[\mathcal{B}_{ij}, \mathbf{CP}_{k-1}]|$ then
35                         $\mathcal{M}_k(i, j) = \mathcal{M}_k[\mathbf{CP}_{k-1}]$
37                     else if $mindeg^-(i, j) > |\mathcal{M}_k[\mathcal{B}_{ij}, \mathbf{CP}_{k-1}]|$ then
39                         $\mathcal{M}_k(i, j) = sum(\mathcal{M}_k[\mathcal{B}_{ij}, \mathbf{CP}_{k-1}]) + \mathcal{M}_k[\mathcal{B}_{ij}^{rem}]$
41                     else
42                         $\mathcal{M}_k(i, j) = \mathcal{M}_k[\mathcal{B}_{ij}]$
44             return $\mathcal{M}_k(i, j)$;
46         return $\mathbf{CP}_k(:, j)$;
48     update $\mathbf{CP}_k$;
50 Opt_FaRS(:, q) = $\mathbf{CP}_L$;
```

Algorithm 4 integrates all the speed-up techniques of Opt_FaRS we just discussed to achieve the optimisation of the FaRS algorithm computation.

Algorithm 4 shows the process of Opt_FaRS with two speed-up approaches clearly. Different lines of Algorithm 4 refer to different underlying formulas and theoretical justifications given in this chapter. The following list shows the references in detail:

- In the **output** line, $\mathbf{CP}_k^\Gamma$ is defined in Theorem 22.
- In line 2, the tracking path $\mathbf{P}$ is described in Definition 8.
- In line 6, the row indexes of $\mathbf{CP}_0$ are determined by the node set **out**, and the column indexes are decided by the in-neighbour nodes set of $\mathbf{P}^L$.
- The speed-up techniques between line 12 to line 15 refer to Lemma 5.
- The optimisation techniques between the line 20 to line 41 refer to Lemma 6.

In summary, for the final role similarity scores of the q's column, not all node pairs in each iteration have an influence on the similarity scores of the q's column. According to this discovery, the Opt_FaRS algorithm can accurately extract the node pairs that have an influence on the role similarity scores of the q's column in each iteration to avoid the calculation of useless information. Leveraging the discussed optimisation techniques, Opt_FaRS greatly reduces the call numbers of the maximum matching algorithm, which is an extremely time-consuming algorithm when applied on large graphs. The most important advantage of the Opt_FaRS algorithm is that it significantly improves the computational efficiency without sacrificing accuracy.

Example 20. *Recall the graph G and the tracking path $\mathbf{P}$ with respect to query d in Figure 5.4. Given that the decay factor is $C = 0.2$, $\lambda = 0.5$, $\Gamma = 1$, iteration number $K = 5$ and the role similarity scores of candidate pool at iteration $k = 4$ is*

$$\mathbf{CP}_4^1 = \begin{array}{c} \\ (b) \\ (c) \end{array} \begin{array}{cccc} (a) & (b) & (c) & (e) \\ \end{array}$$

$$\mathbf{CP}_4^1 = \begin{array}{c|cccc|} & (a) & (b) & (c) & (e) \\ (b) & 0.2 & 0.36 & 0.253 & 0.28 \\ (c) & 0.2 & 0.253 & 0.462 & 0.37 \end{array}^{T}$$

Based on this information, we demonstrate how to compute $\mathbf{CP}_5^1$ *with respect to query d via Eqs.5.19 as follows.*

First, since the in-neighbours of query q is $[b, c]$ *and in-degree of node set* $[b, c]$ *are not equal to zero, query* $d \notin (\mathbf{V}^{(1)}, \mathbf{V}^{(2)})$ *(Definition 10).*

Next, we compute the maximum matching value of the in-neighbour matrix of node pairs $([a, b, c, e], d)$ *row by row.*

1. ***(Row** a.)* *The in-degree of node a is 0, thus* $a \in \boldsymbol{V}^{(1)}$*. According to Lemma 6,* $\mathcal{M}_5^1(a, d) = 0$.

2. ***(Row** b.)* *The in-neighbour of node b is node a and* $a \in \boldsymbol{V}^{(1)}$*, thus* $b \in \boldsymbol{V}^{(2)}$*. According to Lemma 6,* $\mathcal{M}_5^1(b, d) = 1 - C = 0.2$.

3. ***(Row** c.)* *The in-neighbour of node c is node set* $[a, b, e]$*, thus* $c \notin (\mathbf{V}^{(1)}, \mathbf{V}^{(2)})$*. According to Lemma 6, we need calculate the maximum matching of* $\mathbf{CP}_4^2$ *first, and the value has been displayed in Figure 5.6.*

$$\mathbf{CP}_4^1 = \begin{array}{c} \\ b \\ c \end{array}\!\!\left[\begin{array}{cccc} a & b & c & e \\ 0.2 & \boxed{0.36} & 0.253 & 0.28 \\ 0.2 & 0.253 & \boxed{0.462} & 0.37 \end{array}\right]^T$$

Figure 5.6: The Maximum Matching Result of $\mathbf{CP}_4^1$

The bipartite matrix of node pair (c, d) *is* $\mathcal{B}_{cd} = \mathbf{CP}_4^1([a, b, e], [b, c]) = \left[\begin{smallmatrix} 0.2 & 0.36 & 0.28 \\ 0.2 & 0.253 & 0.37 \end{smallmatrix}\right]^T$*. we can see from Figure 5.6,* $\mathcal{M}[\mathcal{B}_{cd}, \mathbf{CP}_4^1]$ *is 1 (the red box in* $\mathbf{CP}_4^1([a, b, e], [b, c])$ *is* $\mathbf{CP}_4^1(b, b) = 0.36$*). Thus, for node pair* (c, d)*,* $|\mathcal{M}[\mathcal{B}_{cd}, \mathbf{CP}_4^1]|$ *is smaller than the* $min(deg_c^-, deg_d^-) = min(3, 2) = 2$*. Based on Lemma 6.(3)* $\mathcal{M}_5^1(c, d) = sum(\mathcal{M}_4[\mathcal{B}_{cd}, \mathbf{CP}_4^1]) + \mathcal{M}_4^1[\mathcal{B}_{cd}^{rem}]0.36 + \mathcal{M}\left[\begin{smallmatrix} 0.2 \\ 0.342 \end{smallmatrix}\right] = 0.702$.

4. ***(Row** d.)* *the number of matching value* $(\mathcal{M}[\mathcal{B}_{dd}, \mathbf{CP}_4^1]) = 2$*, which is equal to* $min(deg_d^-, deg_d^-) = min(2, 2) = 2$*. Thus,* $\mathcal{M}_5^1(d, d) = \mathcal{M}[\mathcal{B}_{dd}, \mathbf{CP}_4^1] = 0.822$.

5. ***(Row** e.)* *after the extract matched the value from the bipartite matrix of node pair* (e, d)*, the only remaining number was* $\mathcal{B}_{ed}^{rem} = [0.2]$*; thus,* $\mathcal{M}_5^1(e, d) = 0.36 + 0.2 = 0.56$.

Finally, bringing the all values of $\mathcal{M}_5^1$ *into Eqs.*(5.19) *to retrieve the role similarity scores of query node d yielded the following:* $\mathbf{CP}_5^1(:, d) = 0.8 \times \left[\begin{smallmatrix} 0 \\ 0.2 \\ 0.73 \\ 0.822 \\ 0.56 \end{smallmatrix}\right] \oslash \left[\begin{smallmatrix} 2 \\ 2 \\ 3 \\ 2 \\ 2 \end{smallmatrix}\right] + \left[\begin{smallmatrix} 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \\ 0.2 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 0.2 \\ 0.28 \\ 0.389 \\ 0.529 \\ 0.424 \end{smallmatrix}\right]$.

Example 20 illustrates the speed-up process in detail. For example, to compute the query role similarity score in the last iteration, the number of maximum matching implemented for Opt_FaRS is only the half of the FaRS algorithm, which greatly speeds up computation without sacrificing any accuracy.

5.5 The Application of **Opt_FaRS** Over Dynamic Graphs

As discussed in Chapter 3, the dynamic graph is a graph subject to a sequence of specific updates; thus, all updates in the graph are known and unique. In real-world graph applications, the dynamic graphs played an increasingly critical role in recent decades. In Chapter 3, we presented D-CoSim, which is a fast and accurate CoSimRank retrieval method on evolving graphs for the node similarity search. The role similarity search over dynamic graphs is also a necessary and urgent topic. However, recent research conducted on similarity search on dynamic graphs is based on the all pairs SimRank algorithm (C. Li et al., 2010) (Yu, Lin, & Zhang, 2014b). Since SimRank is not an applicable role similarity measure, the existing solutions cannot precisely extract role similarity on the graph structure. Furthermore, they are expensive in terms of time and space cost. The Opt_FaRS algorithm we proposed can solve the aforementioned shortcomings efficiently.

According to the basic concept of the single-source Opt_FaRS algorithm 5.19, the role similarity score of each node in a graph is determined by its in-neighbours. Thus, a chunk of the updated edges with the same end node can only change one column of role similarity scores of the old graph similarity score matrix. Therefore, our Opt_FaRS search algorithm can accurately and efficiently retrieve the updated column, without the need of recomputing the whole role similarity score matrix. Furthermore, using the speed-up strategies described in Section 5.4, the Opt_FaRS algorithm can significantly reduce the call numbers of the maximum matching algorithm. At the same time, as a result of the usage of the top Γ^{th} maximum matching of in-neighbour matrix, the Opt_FaRS algorithm can obtain more information from the graph structure and generate more accurate results than the RoleSim algorithm(Eqs. 5.3). These advantages remain when it is applied to dynamic graphs.

The schema for applying the single-source Opt_FaRS algorithm over dynamic graphs has been introduced in detail in this section.

The first step is to partition the updated part of the dynamic graphs. The update of the in-neighbours of a node can change the role similarity score of the corresponding column. The graph partition groups the updated edges that share the same end node.

Having bunched all updated edges into chunks, we update the adjacency matrix of the graph next. Each chunk of the updated edges only changes one column of the adjacency matrix. Note that if the number of nodes in the old graph is smaller than the newly updated graph, the adjacency matrix should be bordered with new zero columns on the right and new zero rows on the bottom. On the opposite side, the adjacency matrix should delete the corresponding column and row from the old adjacency matrix when nodes are deleted from the old graph. Leveraging the new adjacency matrix, the single-source Opt_FaRS algorithm is called for each chunk of the updated edges to retrieve new single-source role similarity scores.

Example 21. *In Figure 5.7, given the old graph G (solid arrows and solid circle nodes) and a sequence of updated edges to G (red dashed arrows and red cross), which is represented by $\widetilde{G}$. Given the query node d, decay factor $C = 0.2$, exact iteration number $L = K = 5$, $\lambda = 0.5$, $\Gamma = 2$. Based on the conception of graph partition, the updated edges of the dynamic graph in Figure 5.7 can be divided into three parts presented on the right side of the figure.*

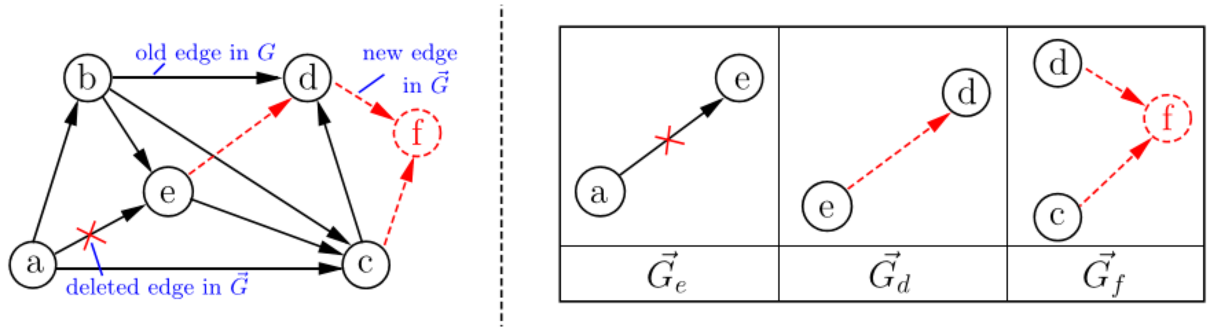


Figure 5.7: Left side: example of old graph G (solid arrows) updated by $\widetilde{G}$ (new edge in dashed arrow and deleted edge with red cross). Right side:the partition of updated edges

We can see from Figure 5.7 that $\widetilde{G} = G_{old} + \widetilde{G}$. The updated edges in Figure 5.4 $\widetilde{G}$ can be written as follows: $G_{new} = \widetilde{G}_e \cup \widetilde{G}_d \cup \widetilde{G}_f$. Having bunched all updated edges into chunks, we update the adjacency matrix of the new graph chunk by chunk. Based on the new adjacency matrix, recall the Opt_FaRS algorithm to update the role similarity scores with respect to query d.

5.6 Experimental Evaluation

This section shows an experimental study on real datasets to compare our algorithms (FaRS and Opt_FaRS) with other baseline algorithms. The experimental results verified the superiority of the FaRS and Opt_FaRS algorithms based on graph topology.

The performance efficiency is evaluated using three metrics.

- The first metric is the impact of different coefficient choices on the accuracy of the FaRS algorithm.
- The second one is accuracy. FaRS can get more information from the structure of graphs. The role similarity results of FaRS are more accurate than other algorithms. Opt_FaRS is an optimisation algorithm based on the FaRS algorithm, and the results of Opt_FaRS are precisely the same as the results of the FaRS algorithm.
- The last one was time efficiency. FaRS quickly answers the role similarity search with respect to the query based on graph topology. Opt_FaRS is much faster than the FaRS algorithm without loss accuracy.

5.6.1 Experimental Settings

Datasets. We evaluate our algorithm over a group of real-life datasets. We used the following public datasets:

email-Eu-core-temporal (EU)[1]. Email data from a prominent European research organisation is used to create the network. All incoming and outgoing e-mail data between members of the research institution is anonymised in the network. In a research institute, every employee can be considered a node. A directed edge u to v appears when employee u sends an email to employee v. If there is already a directed edge between the node pair (u, v), multiple emails sent between two employees and reverse emails will no longer generate edges between node pair (u, v) to ensure the uniqueness and stability of the graph structure. The graph structure provides a separate directed edge for each receiver when an email is sent to numerous employees simultaneously. Emails reflect only the internal communications within the organisation, and the dataset does not include incoming or outgoing messages from other regions of the globe.

[1]https://snap.stanford.edu/index.html

This organisation is mainly composed of four departments. The mail exchanges between members of the four departments in the organisation are represented by four separate networks. The composition of the four networks is the same as the organisation's mail exchange network. For a member of the organisation, the corresponding node index of the organisation's network is different from the index in the departmental network.

The size of each dataset has been illustrated in Table 5.1.

Datasets		**#-Nodes**	**#-Edges**	**Type**
email-Eu-core-temporal	(EU)	986	24,929	Directed
Department 1	(Dept-1)	309	3,031	Directed
Department 2	(Dept-2)	162	1,772	Directed
Department 3	(Dept-3)	89	1,506	Directed
Department 4	(Dept-4)	142	1,375	Directed

Table 5.1: Description of the Five Datasets

Compared Algorithms. We implemente FaRS and Opt_FaRS over the five real-life datasets, respectively, and compare them with two state-of-the-art similarity search competitors and another format of FaRS:

(a) CSR, a method developed by (Rothe & Schütze, 2014) that retrieves a CoSimRank score from the sum of the dot product of two Personalised PageRank vectors;

(b) RoleSim (Rothe & Schütze, 2014), a state-of-the-art role similarity search algorithm, which generates role similarity scores based on the average value of maximum matching.

(c) FaRS_N, which is another format of the FaRS algorithm that generates role similarity scores by computing the average maximum matching of the remaining in-neighbour matrix rather than top Γ maximum matching(FaRS).

Parameters. We chose the following parameters according to the results of parameter evaluation in Section 5.6.2:

(a) the decay factor $C = 0.2$,

(b) the number of iterations $K = 5$,

(c) the order of maximum matching $\Gamma = 3$,

(d) the relative weight $\lambda = 0.7$.

Evaluation Metrics. To evaluate role similarity ranking results on real-life datasets, we used k-means clustering (Arthur & Vassilvitskii, 2006; Lloyd, 1982; Bock, 2007). K-means clustering is a vector quantisation technique that seeks to divide n observations into k clusters, with each observation belonging to the cluster with the closest mean (cluster centres or cluster centroid), which serves as the cluster's prototype[2].

For ground truth, first, we calculated the role similarity scores matrix using our algorithm and other baseline algorithms, respectively. Then, k-means clustering was done on different role similarity scores matrices to divide the data into several groups. According to the characteristics of the k-means clustering, the role similarity scores between the nodes in each group was higher. Next, given different queries, the top 20 nodes most similar to the query node in each algorithm are extracted. At the same time, we identify the group that the query node belongs to, and the group is divided by k-means clustering. Finally, we need to calculate how many of the 20 nodes (most similar to the query of each algorithm) are the same as the nodes of the query group. The higher the overlap ratio, the higher is the accuracy.

All experiments are conducted on a PC with Intel Core i7-6700 3.40GHz CPU and 64GB memory using Windows 10. Each experiment is repeated five times, with the average results being shown here.

5.6.2 Experimental Results

In this section, we display the experimental results of comparing our algorithm with other baseline algorithms over five real-life datasets. The experimental results comprise three parts: parameter evaluation, accuracy and time efficiency. We first implemente FaRS over the real-life dataset to illustrate the better choice of parameter.

Parameter Evaluation. We evaluate two parameters of the FaRS algorithm in this part, λ and Γ. The method involved implementing our algorithm over the Dept-3 dataset and comparing the algorithm's accuracy with different parameter values. For accuracy evaluation, we generated the role similarity score matrix of the graph first. Then, we used the k-means clustering (Arthur & Vassilvitskii, 2006; Lloyd, 1982; Bock, 2007) method to evaluate the accuracy of our algorithm for different parameter values.

Figure 5.8 depicts the accuracy of the FaRS algorithm in the case of different λ values.

[2]https://en.wikipedia.org/wiki

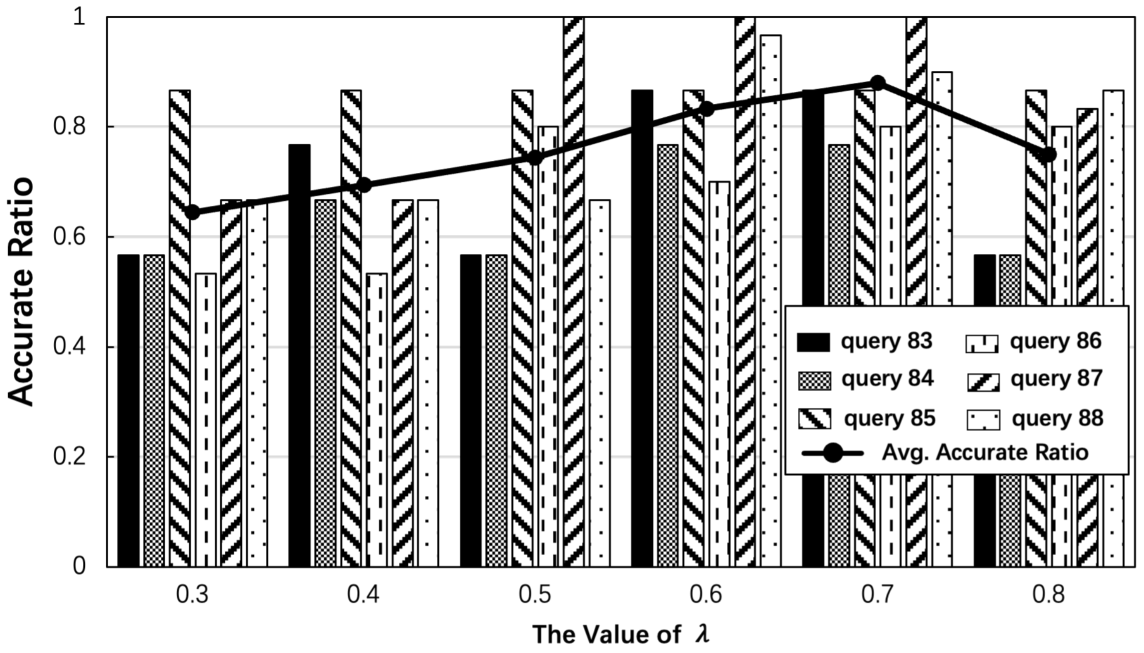


Figure 5.8: FaRS with different λ

We implemente the FaRS algorithm over the Dept-3 dataset. The ordinate in Figure 5.8 is the accuracy ratio. The accuracy ratio is determined by the number of duplicate nodes in the combination of two nodes sets. A combination is composed of the top 20 nodes with the highest role similarity scores to the query node. Another combination is composed of nodes in the same k-means group as the query node. Thus, when more nodes belong to the two node sets at the same time, the algorithm is more accurate. We select six different λ values ($\lambda = [0.3, 0.4, 0.5, 0.6, 0.7, 0.8]$), and six query nodes ($Q = [83, 84, 85, 86, 87, 88]$) to evaluate which λ value is more appropriate. From Figure 5.8, we can see that when $\lambda = [0.6, 0.7]$, the accurate ratio is relatively high. Especially for query 87, the 20 nodes most similar to query node 87 belong to the same k-means group with the query. The line chart in Figure 5.8 shows the average accurate ratio with respect to different λ values, respectively. When $\lambda = 0.7$, the accurate ratio of the FaRS algorithm is relatively highest. So, in future experiments, the value of $\lambda = 0.7$.

Figure 5.9 depicts the accuracy of the FaRS algorithm in the case of different Γ values. We implemente the FaRS algorithm over the Dept-3 dataset.

In the experiment, we chose five different ($\Gamma = [2, 3, 4, 5, 6]$) and the query set $Q = [70.71, 72, 73, 74]$.

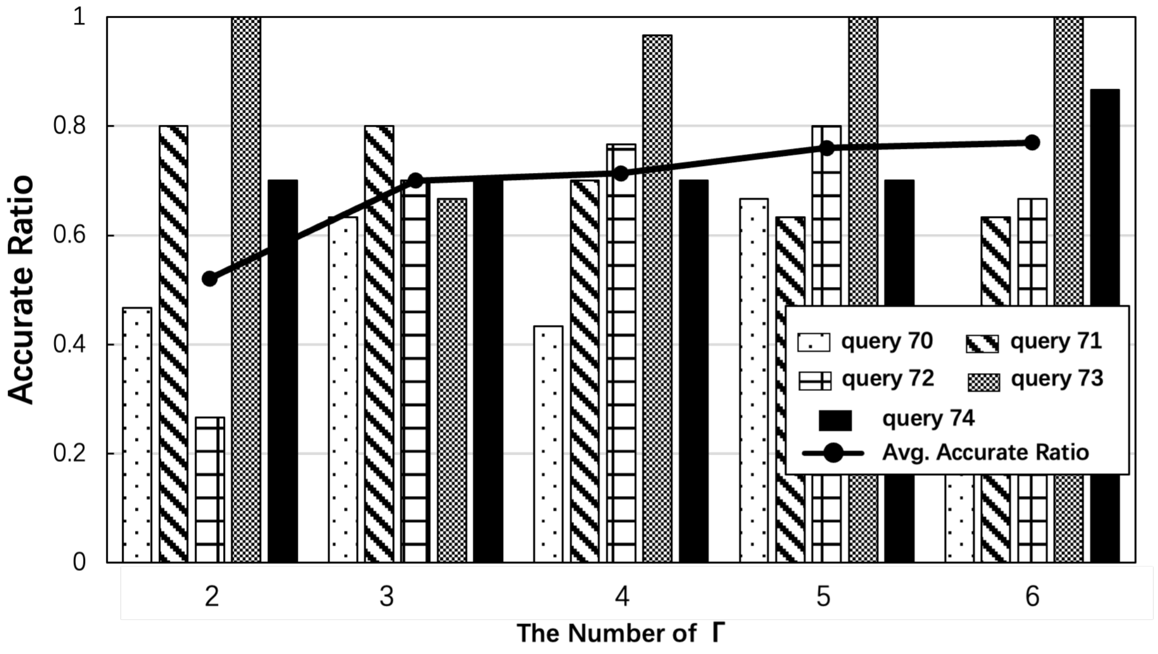


Figure 5.9: FaRS with different Γ

In the FaRS algorithm, Γ represents that the algorithm needs to calculate the top Γ maximum matching values for the role similarity score. Since the grouping of the k-means clustering group will be different each time, the accuracy of each Γ value for each query is the average value after five trials. In Figure 5.9, the bar chart shows the accuracy of the different queries using different Γ values in the FaRS algorithm. The line chart is the average of the accurate ratio of five queries corresponding to a Γ value. It can be seen from the trend of the line chart that as the Γ value increases, the accuracy of the algorithm improves. It is worth mentioning that the accuracy is greatly improved when $\Gamma = 3$. After $\Gamma = 3$, the increase in accuracy tends to be flat. Therefore, the algorithm performs best when $\Gamma = 3$ (high accuracy and less time-consuming).

Accurate Evaluation. We next compare the accurate ratio of our algorithms (FaRS and Opt_FaRS) and the other baseline algorithms over six real-life datasets. In this part, we used two methods to evaluate the accuracy of the algorithms. The first method is the k-means clustering mentioned earlier. The second method is to evaluate the accuracy according to the characteristics of the graph itself. EU is the mail communication network of the employees (core) of the organisation, which includes the mail communication network of the four departments. Based on the structure of EU, the role similarity between employees in each department is higher than that between employees in different departments. Therefore, EU is divided into five parts, including department 1 (Dept-1), department 2 (Dept-2), department 3 (Dept-3), department 4 (Dept-4) and the remaining employees. We randomly selecte a node in each part as the query and then tested the number of nodes in the corresponding department among the first 20 nodes that are similar to the query in different algorithms. This is an important reason why we select these datasets to evaluate our algorithms.

First, we use the k-means clustering to test the accuracy of the algorithms on the mail exchange network of the four departments, respectively. For the k-means clustering of each algorithm, the partition number $k = 6$. The length of order list of each algorithm with respect to the query is 20.

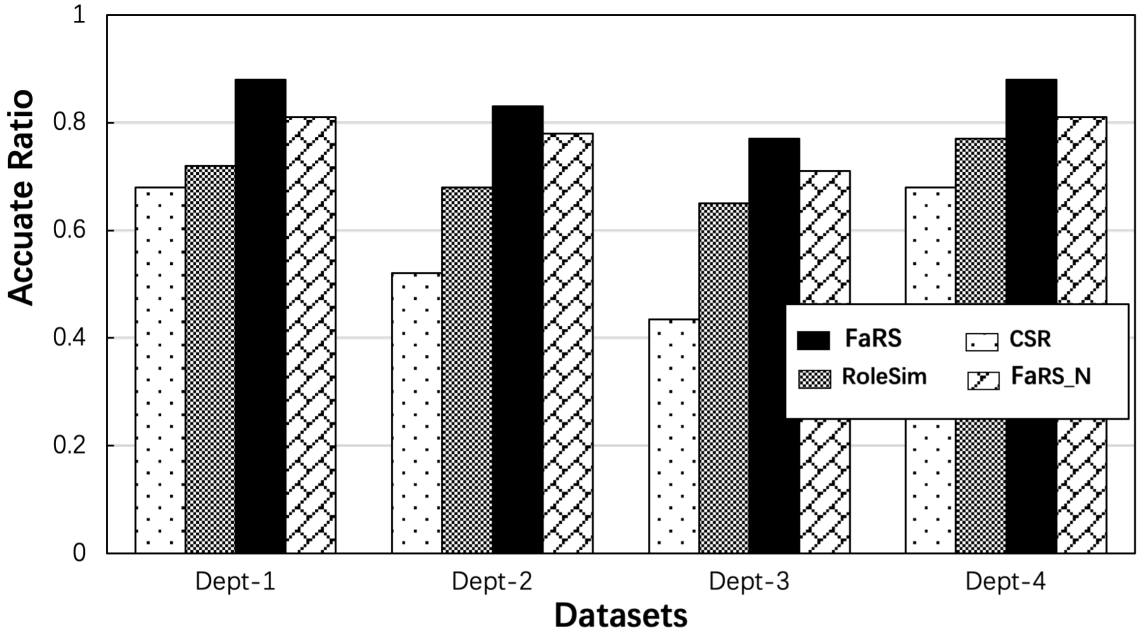


Figure 5.10: Accurate Evaluation of Algorithms Over Four Departments' Networks

Figure 5.10 shows the accurate ratio of different algorithms on four datasets. The accurate ratio of the role similarity search of the CSR algorithm is relatively low on each dataset. Next, the role similarity search accuracy of the RoleSim algorithm is higher than that of the CSR algorithm but lower than that of the FaRS algorithm. The role similarity detection accuracy of the FaRS algorithm outperforms the four datasets. The accuracy of the FaRS_N algorithm is better than that of the RoleSim and CSR algorithms, but it is not as good as that of the FaRS algorithm. So, our algorithm uses the top Γ maximum matching method instead of removing the value that has been first-order maximum matched and finding the maximum matching value of the remaining matrix to calculate the role similarity scores.

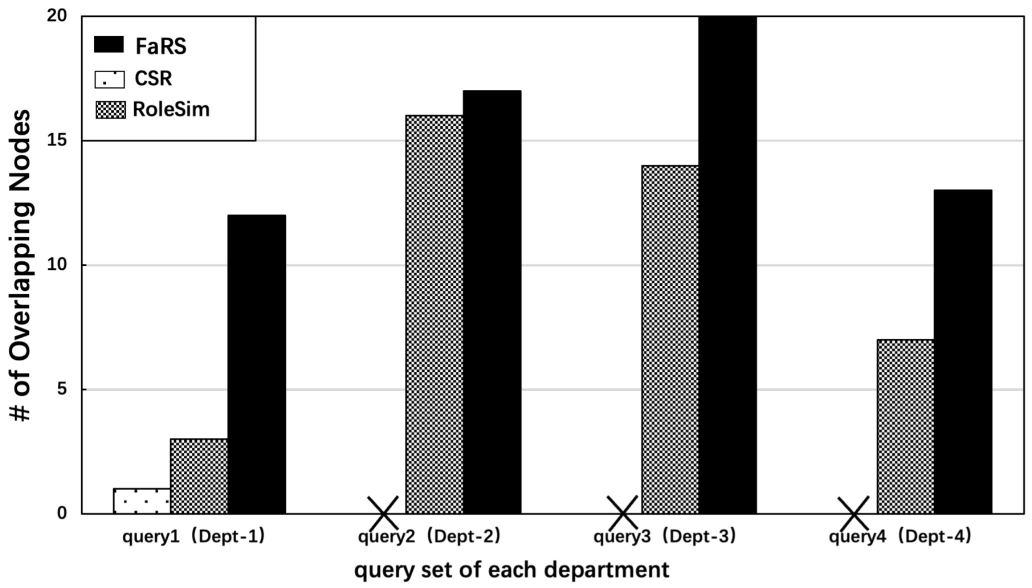


Figure 5.11: Accurate Evaluation of the Opt_FaRS Algorithm on EU(K-means Clustering Method)

Next, we evaluate the accuracy of our algorithm and other baseline algorithms over the EU dataset. We randomly selecte four nodes from the four departments as the query set of the corresponding department. Then, after ranking the top 20 nodes, each algorithm is found to have the highest role similarity scores to the query. Finally, we test the top 20 nodes of each algorithm with respect to each query and how many nodes belonged to the query's corresponding department. The higher the degree of coincidence, the higher is the accuracy of the algorithm. Figure 5.11 shows that the FaRS algorithm has always maintained a high degree of accuracy. The second is the RoleSim algorithm. The accuracy of role similarity search of the CSR algorithm is always been relatively lower. To sum up the information from Figure 5.10 and Figure 5.11, the FaRS algorithm outperformed the best-known algorithms CSR and RoleSim over the five real-life datasets.

Finally, we evaluate the accuracy of Opt_FaRS over the EU dataset. The Opt_FaRS algorithm is an accelerated algorithm based on the FaRS algorithm. Thus, we evaluate the accuracy of Opt_FaRS relative to the FaRS on Dept-4. We randomly pick up various query sets with its size $|Q|$ varying from 10 to 30. For each query set Q, based on the role similarity scores from Opt_FaRS, we measure their similarity ranking results via NDCG (Normalised Discounted Cumulative Gain) (Y. Wang et al., 2013)(details in Section 3.4.2.1). Thus, NDCG = 1 implies that the role similarity ranking of the compared algorithm perfectly matches that of FaRS, with no loss in accuracy.

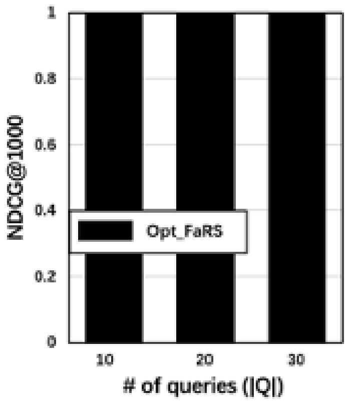


Figure 5.12: Accurate Evaluation of the Opt_FaRS Algorithm (NDCG Method)

From Figure 5.12, we notice that for each query set Q, the NDCGs of Opt_FaRS was 1s, implying that Opt_FaRS does not sacrifice any accuracy for their speedup. This verify the correctness of Lemma 6.

Time Efficiency. We evaluate the time efficiency of our algorithms (FaRS and Opt_FaRS) and the baseline algorithms over five networks.

Figure 5.13 depicts the time efficiency of our algorithms (FaRS and Opt_FaRS) and other baseline algorithms over five different realistic networks . The elapsed time comes from the computation of single-source role similarity scores for each query. For each dataset, we randomly take $|Q| = 20$ queries. The FaRS and RoleSim algorithms have comparable time efficiency on five datasets. The CSR algorithm costs relatively less time, but the accuracy of the algorithm for calculating the role similarity scores is not high. The Opt_FaRS algorithm has great efficiency, which is more than the FaRS and RoleSim algorithm over five real-life datasets.

5.7 Related Work

Similarity detection research based on graph structure has been a trendy research topic in the past decades. Among the research on similarity search over graphs, SimRank (Jeh & Widom, 2002) has been proposed as the most fundamental similarity search algorithm. The basic intuition of the algorithm is that "two objects are similar if they are related to similar

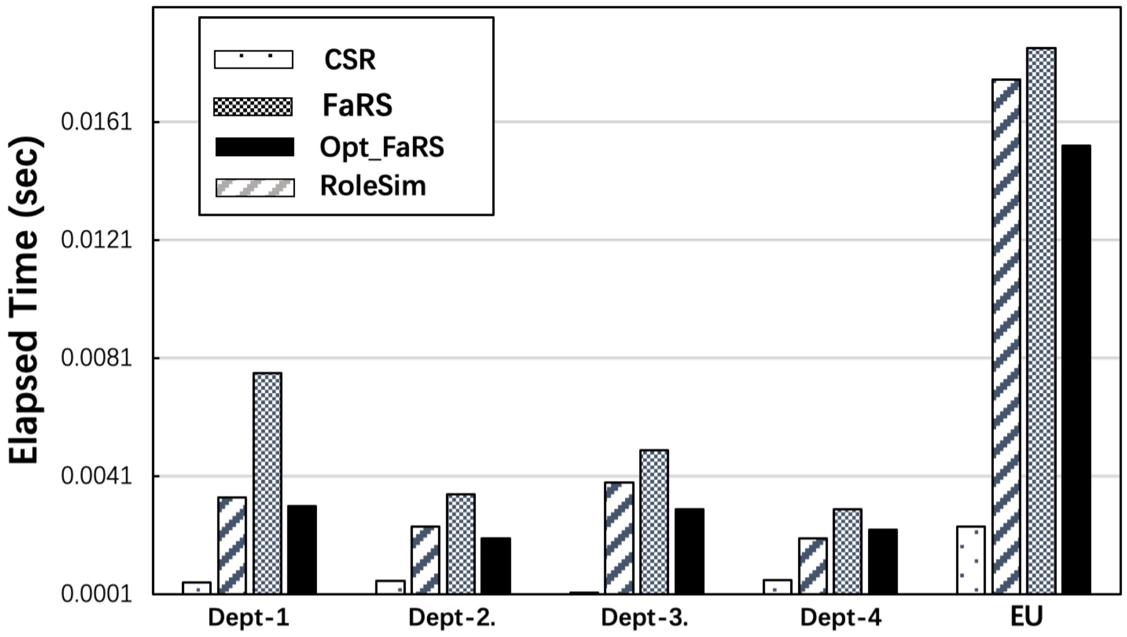


Figure 5.13: Time Efficiency of Algorithms

objects". Afterwards, many SimRank-based algorithms have been proposed((Kusumoto et al., 2014; C. Li et al., 2010; Antonellis, Molina, & Chang, 2008; He, Feng, Li, & Chen, 2010; P. Li, Liu, Yu, He, & Du, 2010; Lizorkin, Velikhov, Grinev, & Turdakov, 2008b; Ma, Lin, & Lin, 2011)). Similarity detection based on graph structure can help people live more efficiently and conveniently (J. Wu et al., 2021). For example, based on the anonymous online shopping network generated by the user's shopping record, the similarity detection on the online shopping network can automatically generate a recommendation list. This list can greatly increase the efficiency of customers in choosing products among the huge number of online products. At the same time, it can also help merchants to reasonably produce and reserve commodities to avoid waste.

With the continuous development of similarity detection algorithms on graph structures, more and more researches have begun to focus on semantic similarity detection on graph structures((Everett, 1985; Yu, Iranmanesh, Haldar, Zhang, & Ferhatosmanoglu, 2020; Lee, 2012; Shao et al., 2019; Lin et al., 2012; Milo, Somech, & Youngmann, 2019; Chen et al., 2021; Rothe & Schütze, 2014)). Recall the previous example of the online shopping recommendation system. The similarity detection algorithm can accurately recommend a product list that is closest to a product. However, role similarity detection can recommend related products based on the user's individual characteristics such as gender, occupation etc. Among the many role similarity detection algorithms, the RoleSim (Rothe & Schütze, 2014) algorithm stands out. Its basic intuition is that "two nodes have the similar role if they interact with equivalent sets of neighbours". The RoleSim algorithm calculates the role similarity value based on the maximum matching value of the in-neighbours role similarity matrix of the node pair, rather than the average value of the in-neighbours similarity matrix((Jeh & Widom, 2002)). The maximum matching algorithm can only extract the information of some node pairs in the matrix. Especially when the in-degrees of the two nodes are different, part of the in-neighbours information will be directly ignored. But our algorithm can well capture the information ignored by the RoleSim algorithm to achieve a more accurate role similarity detection value. At the same time, an accelerated algorithm is proposed based on the RoleSim algorithm. A threshold is given in the acceleration algorithm. When calculating the node pair role similarity value, all the in-neighbours similarity value matrix of the node pairs are no longer used for maximum matching to calculate the role similarity value, but the in-neighbours similarity value greater than the threshold can be used for calculation.

Some related researches are derived based on the RoleSim algorithm((Shao et al., 2019; Chen et al., 2021; Chen, Lai, Qin, & Lin, 2020)). A seedless de-anonymisation method called RoleMatch was proposed by (Shao et al., 2019). The RoleMatch algorithm can be divided into two parts. The first part is the novel role similarity detection algorithm RoleSim++. RoleSim++ calculates the role similarity value of a node pair based on the maximum matching value extracted from the in-neighbours and out-neighbours role similarity value matrix of the node pair, which is different from the RoleSim algorithm. In order to improve the computational efficiency of the RoleSim++ algorithm, the α-RoleSim++ algorithm was proposed. The α-RoleSim++ algorithm gives a threshold value and then only extracts information from the node pairs' role similarity scores greater than the threshold, and other node pairs are ignored. Then, based on the calculated role similarity score, the NeighborMatch matching algorithm was proposed to find a good mapping between the anonymised network

The most state-of-the-art algorithm for role similarity search is StructSim (Chen et al., 2020). StructSim calculates the role similarity scores through the maximum matching value of the horizontal similarity between each k-neighborhood subgraph. In order to improve the computational efficiency of the StructSim algorithm, the maximum match in the algorithm is replaced with BinCount match. In the BinCount matching algorithm, the index of the nodes of each layer needs to be recorded. Flajolet-Martin Sketch was proposed to create the index of the nodes of each layer more efficiently.

5.8 Conclusion

This chapter propose an accurate role similarity search algorithm FaRS based on graph topology. We also present an accelerate algorithm, Opt_FaRS, based on FaRS. First, the FaRS algorithm uses the top Γ maximum matching values to calculate single-source role similarity scores. Thus, it can capture more information from node-pair in-neighbour role similarity scores matrix than the RoleSim algorithm, which guarantees the accuracy of FaRS algorithm. We also prove the convergence, uniqueness, symmetry, boundedness and triangular inequality of the FaRS algorithm. Then, we propose an acceleration algorithm based on the FaRS algorithm, Opt_FaRS. The acceleration algorithm process as follows: (a) starting from the query node, we record the multi-hop backward tracking path by recording the in-neighbours index of nodes. (b) filter **out** nodes set with all nodes whose out-degree is greater than zero. (c) calculate the role similarity values of each level of candidate pool through the p-speedup and out-speedup acceleration approaches until convergence. The column index of the candidate pool is determined by the tracking path index of the corresponding level. The row index of all candidate pools is determined by the **out** nodes set. Finally, we implement the Opt_FaRS algorithm over dynamic graphs. We evaluate our algorithms and the baseline algorithms on five real datasets. The experimental results show that the FaRS algorithm obtains a more accurate role similarity value than the baselines algorithms. At the same time, the Opt_FaRS algorithm dramatically improves the calculation speed of the FaRS algorithm without losing accuracy.

6 Conclusions and Future Work

This thesis study several efficient techniques for assessing link-based single-source relevance on large-scale static and dynamic networks (graphs), including incremental D-CoSim and D-deCoSim algorithms on large dynamic networks, F-CoSim, Opt_F-CoSim and F-CoSim_Para on large-scale static networks and the development of a more accurate role relevance scoring algorithm for enriching semantics. In Section 6.1 below, we outline the thesis' primary contributions, and in Section 6.2, we discuss general avenues for further research stemming from this thesis.

6.1 Thesis Summary

Chapter 1 highlight the need for efficiently assessing single-source relevance through motivating applications as well as the challenges and contributions of this thesis. In Chapter 2, we provide a comprehensive literature review of graph-based similarity search according to the different types (iterative algorithms, Monte Carlo sampling, matrix-based methods, static algorithms, dynamic algorithms etc.). The main technical contributions of the thesis have been explained in Chapters 3–5.

- **Fast and Accurate Similarity Search Over Evolving Graphs.** Chapter 3 propose efficient methods for conducting similarity search on large-scale evolving networks. (1) For large incremental dynamic graphs, a fast and accurate similarity search algorithm, the D-CoSim algorithm, is presented to generate the similarity scores over incremental dynamic graphs. (2) For large decremental dynamic graphs, a novel similarity search algorithm over decremental dynamic graphs, the D-deCoSim algorithm, is proposed as well. The D-deCoSim algorithm can be divided into two parts: D-deCoSim(Node) over dynamic graphs with node deletion and D-deCoSim(Edge) over dynamic graphs with edge deletion. Experiments demonstrate that our algorithm D-CoSim and D-deCoSim steadily outperform two state-of-the-art CoSimRank competitors (CSR (Rothe & Schütze, 2014) and CSM (Yu & McCann, 2015a)) with 3-5 order-of-magnitude(time efficiency) on real large-scale datasets. Compared with CoSimRank and CSM algorithms, our algorithms show a great advantage in memory efficiency. In particular, CSM cannot be implemented for large graphs since its memory complexity is $O(n^2)$. The experimental results also show that our algorithms do not compromise on accuracy while increasing the (search) speed.

- **Scalability Similarity Search Over Large-Scale Graphs.** Chapter 4 presente an innovative paradigm that supports similarity search over large-scale static graphs. (1) The single-source F-CoSim algorithm can efficiently retrieve similarity scores by (a) finding a "spanning polytree" from the graph; (b) designing an efficient algorithm to retrieve the CoSimRank scores over a spanning polytree; and (c) applying D-CoSim to evaluate the changes in response to the difference between the spanning polytree and the graph. (2) To accelerate the F-CoSim algorithm, an effective pruning technique is also leveraged Opt_F-CoSim. Opt_F-CoSim includes three techniques: (a) Opt_Find_Spanning_Polytree; (b) Single-source similarity search algorithm over spanning polytree; (c) Inspired by parallel computing, we propose the novel and fast similarity search algorithm F-CoSim-Para based on graph topology. We conduct extensive experiments on real and large datasets, which demonstrate that our efficient similarity search algorithms, F-CoSim and Opt_F-CoSim, outperforms state-of-the-art approaches on static graphs with a speed-up of up to 9.8 times, and the proposed algorithms shows great advantage on memory efficiency to CSM. The more important thing is that the proposed algorithms do not compromise on accuracy for speed.

- **An Accurate Role Similarity Search Over Networks.** Chapter 5 present a more accurate paradigm that supports role similarity search over social networks. For static graphs, the FaRS algorithm uses the top Γ maximum matching values to calculate single-source role similarity scores based on graph topology. Additionally, an accelerated algorithm with two efficient techniques is devised for speeding up FaRS over social networks, and it is named Opt_FaRS. For dynamic graphs, we propose a diagram to implement Opt_FaRS over dynamic graphs. Experiments demonstrate that our FaRS and Opt_FaRS algorithms are steadily becoming more accurate than the two state-of-the-art CoSimRank competitors (CSR (Rothe & Schütze, 2014) and RS (Rothe & Schütze, 2014)) on real datasets. The experimental results also show that our algorithm Opt_FaRS does not scarify accuracy for speeding up.

6.2 Future Avenues

Aside from the specific unresolved questions discussed in individual chapters, there are other broad directions in which the work described in this thesis could be expanded.

In the big data area, large-scale highly-interconnected graphs pervade our lives. A critical task in graph mining is to evaluate node importance based on the link structures. Since the invention of PageRank (Google, 2011), several methods have been proposed for ranking nodes, as it has essential applications in social community detection (Chamberlain, Levy-Kramer, Humby, & Deisenroth, 2018), citation analysis (Elleby & Ingwersen, 2010) and recommend systems (Machado et al., 2014). Among them, Personalised PageRank (PPR) is an appealing tool for measuring node importance (Rothe & Schütze, 2014).

In contrast to PageRank, which distributes the starting nodes uniformly, PPR walks are biased towards personal interests. PPR scores can be interpreted by "random surfers". Given a set of preference nodes P, a surfer can either jump to an outgoing neighbour with a probability of $(1 - \alpha)$ or teleport back to a node in P with a probability of α, where α is the teleport probability. This procedure is performed repeatedly until it converges into a steady state. The final distribution of the random surfers on nodes is the PPR scores with respect to P.

Most previous studies on PPR (Lofgren, Banerjee, Goel, & Seshadhri, 2014) assumed that the underlying graph is deterministic. However, many real graphs are often noisy and uncertain due to various reasons, such as the lack of precise information needs, noisy measurements or explicit manipulation for privacy purposes. To represent uncertainty in noisy graph data, unlike traditional studies (Potamias, Bonchi, Gionis, & Kollios, 2010; J. H. Kim, 2017) that assumed the existence probabilities of individual edges to be independent of each other, the state-of-the-art literature (J. H. Kim, Li, Candan, & Sapino, 2017) proposed another powerful uncertain graph model and considered the PPR search problem on this type of uncertainty, where the existence probabilities within bunches of uncertain edges are mutually dependent. This uncertainty model is quite useful when one is aware of the existence of an edge but has no idea between which pairs of nodes the edge exists. For instance, in a social network, we can deduce that one of the several friends (a set of target nodes) of an individual (a source node) has committed a crime, but we do not know which friend. Another example of named entity disambiguation could be an instance where one may be aware that a name (a source node) referred to in a Wikipedia page is one of the many named entities (a set of target nodes) in a knowledge base, but one does not identify which one is the correct entity. Therefore, uncertain graphs modelled by (J. H. Kim et al., 2017) have many emerging real applications.

Although the uncertain model proposed by (J. H. Kim et al., 2017) is effective, it poses striking challenges for efficient PPR search over certain graphs. A naive way to evaluate PPR scores over an uncertain graph G is to enumerate all the possible worlds of G and average out the PPR scores over each possible world, which is prohibitively expensive. To speed up the retrieval, the best-of-breed research (J. H. Kim et al., 2017) proposed a fast scheme, uPPR. It approximates PPR on uncertain graphs at the expense of accuracy. However, uPPR suffers from two limitations: (i) uPPR does not always guarantee high accuracy, as it approximates PPR to only the first order with respect to uncertain edges while neglecting all high-order terms. (ii) uPPR is not scalable on large uncertain graphs, as it involves a costly pre-computation phase to materialise the inverse of block matrices for a certain part of the graphs.

Therefore, similarity search over uncertain graphs plays a key role in ranking the objects of people's lives. However, existing works have limitations on accuracy and scalability, which can be integrated as an interesting direction of future work.

References

Akoglu, L., & Faloutsos, C. (2009). RTG: a recursive realistic graph generator using random typing. *Data Min. Knowl. Discov.*, *19*(2), 194–209. Retrieved from `https://doi.org/10.1007/s10618-009-0140-7` doi: 10.1007/s10618-009-0140-7

Almasi, G. S., & Gottlieb, A. (1994). *Highly parallel computing.* Benjamin-Cummings Publishing Co., Inc.

Antonellis, I., Garcia-Molina, H., & Chang, C.-C. (2008). Simrank++ query rewriting through link analysis of the clickgraph (poster). In *Proceedings of the 17th international conference on world wide web* (pp. 1177–1178).

Antonellis, I., Molina, H. G., & Chang, C. (2008). SimRank++: Query rewriting through link analysis of the click graph. *PVLDB*, *1*, 408–421.

Arthur, D., & Vassilvitskii, S. (2006). *k-means++: The advantages of careful seeding* (Tech. Rep.). Stanford.

Bahmani, B., Chowdhury, A., & Goel, A. (2010). Fast incremental and personalized PageRank. *PVLDB*, *4*(3).

Bahmani, B., Kumar, R., Mahdian, M., & Upfal, E. (2012). Pagerank on an evolving graph. In *Proceedings of the 18th acm sigkdd international conference on knowledge discovery and data mining* (pp. 24–32).

Batagelj, V., Doreian, P., & Ferligoj, A. (1992). An optimizational approach to regular equivalence. *Social networks*, *14*(1-2), 121–135.

Berberich, K., Bedathur, S., Weikum, G., & Vazirgiannis, M. (2007). Comparing apples and oranges: normalized pagerank for evolving graphs. In *Proceedings of the 16th international conference on world wide web* (pp. 1145–1146).

Biehn, S. E., & Lindert, S. (2021). Accurate protein structure prediction with hydroxyl radical protein footprinting data. *Nature communications*, *12*(1), 1–10.

Bock, H.-H. (2007). Clustering methods: a history of k-means algorithms. *Selected contributions in data analysis and classification*, 161–172.

Borgatti, S. P., & Everett, M. G. (1993). Two algorithms for computing regular equivalence. *Social networks*, *15*(4), 361–376.

Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual web search engine.

Computer networks and ISDN systems, *30*(1-7), 107–117.

Burkard, R., Dell'Amico, M., & Martello, S. (2012). *Assignment problems: revised reprint.* SIAM.

Buyukkokten, O., Garcia-Molina, H., & Paepcke, A. (2001). Seeing the whole in parts: text summarization for web browsing on handheld devices. In *Proceedings of the 10th international conference on world wide web* (pp. 652–662).

Cannistraci, C. V., Alanis-Lobato, G., & Ravasi, T. (2013). From link-prediction in brain connectomes and protein interactomes to the local-community-paradigm in complex networks. *Scientific reports*, *3*(1), 1–14.

Catherine, R., & Cohen, W. (2016). Personalized recommendations using knowledge graphs: A probabilistic logic programming approach. In *Proceedings of the 10th acm conference on recommender systems* (pp. 325–332).

Chamberlain, B. P., Levy-Kramer, J., Humby, C., & Deisenroth, M. P. (2018). Real-time community detection in full social networks on a laptop. *PloS one*, *13*(1), e0188702.

Chaudhuri, S., Ganti, V., & Xin, D. (2009). Exploiting web search to generate synonyms for entities. In *Proceedings of the 18th international conference on world wide web* (pp. 151–160).

Chen, X., Lai, L., Qin, L., & Lin, X. (2020). Structsim: Querying structural node similarity at billion scale. In *2020 ieee 36th international conference on data engineering (icde)* (pp. 1950–1953).

Chen, X., Lai, L., Qin, L., & Lin, X. (2021). Efficient structural node similarity computation on billion-scale graphs. *The VLDB Journal*, *30*(3), 471–493.

Cho, J., Garcia-Molina, H., & Page, L. (1998). Efficient crawling through url ordering. *Computer Networks and ISDN Systems*, *30*(1-7), 161–172.

Colliri, T., & Zhao, L. (2020). A network-based approach to predict new affected regions and the spread evolution of covid-19. *Available at SSRN 3577663*.

da Silva Villaca, R., de Paula, L. B., Pasquini, R., & Magalhaes, M. F. (2013). A similarity search system based on the hamming distance of social profiles. In *2013 ieee seventh international conference on semantic computing* (pp. 90–93).

Desikan, P., Pathak, N., Srivastava, J., & Kumar, V. (2005). Incremental page rank computation on evolving graphs. In *Special interest tracks and posters of the 14th international* *conference on world wide web* (pp. 1094–1095).

Diao, L., Wang, H., Alsarra, S., Yen, I.-L., & Bastani, F. (2019). A smart role mapping recommendation system. In *2019 ieee 43rd annual computer software and applications conference (compsac)* (Vol. 2, pp. 135–140).

Du, H. (2010). Chemical molecules search based on graph similarity measure. In *2010 international conference on biomedical engineering and computer science* (pp. 1–4).

Elleby, A., & Ingwersen, P. (2010). Publication point indicators: A comparative case study of two publication point systems and citation impact in an interdisciplinary context. *Journal of Informetrics*, *4*(4), 512–523.

Eto, M. (2019). Extended co-citation search: Graph-based document retrieval on a co-citation network containing citation context information. *Information Processing & Management*, *56*(6), 102046.

Everett, M. G. (1985). Role similarity and complexity in social networks. *Social Networks*, *7*(4), 353–359.

Fogaras, D., & Rácz, B. (2005). Scaling link-based similarity search. In *Proceedings of the 14th international conference on world wide web* (pp. 641–650).

Fujiwara, Y., Nakatsuji, M., Shiokawa, H., & Onizuka, M. (2013). Efficient search algorithm for simrank. In *Data engineering (icde), 2013 ieee 29th international conference on* (pp. 589–600).

Gabow, H. N., Kaplan, H., & Tarjan, R. E. (2001). Unique maximum matching algorithms. *Journal of Algorithms*, *40*(2), 159–183.

Garg, S., Gupta, T., Carlsson, N., & Mahanti, A. (2009). Evolution of an online social aggregation network: an empirical study. In *Proceedings of the 9th acm sigcomm conference on internet measurement* (pp. 315–321).

Google. (2011). Facts about google and competition.

Gravano, L., Garcia-Molina, H., & Tomasic, A. (1994). The effectiveness of gioss for the text database discovery problem. In *Acm sigmod record* (Vol. 23, pp. 126–137).

Gross, J. L., & Tucker, T. W. (2009). *Topics in topological graph theory* (Vol. 128). Cambridge University Press.

Gross, J. L., Yellen, J., & Anderson, M. (2018). *Graph theory and its applications*. Chapman and Hall/CRC.

Gupta, S., Khodabakhsh, A., Mortagy, H., & Nikolova, E. (2021). Electrical flows over spanning trees. *Mathematical Programming*, 1–41.

Hang, C.-W., & Singh, M. P. (2010). Trust-based recommendation based on graph similarity. In *Proceedings of the 13th international workshop on trust in agent societies (trust). toronto, canada* (Vol. 82).

Haveliwala, T. H. (2003). Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search. *IEEE transactions on knowledge and data engineering*, *15*(4), 784–796.

He, G., Feng, H., Li, C., & Chen, H. (2010). Parallel SimRank computation on large graphs with iterative aggregation. In *Kdd.*

Jeh, G., & Widom, J. (2002). Simrank: a measure of structural-context similarity. In *Proceedings of the eighth acm sigkdd international conference on knowledge discovery and data mining* (pp. 538–543).

Jeh, G., & Widom, J. (2003). Scaling personalized web search. In *Proceedings of the 12th international conference on world wide web* (pp. 271–279).

Jiang, M., Fu, A. W., Wong, R. C., & Wang, K. (2017). READS: A random walk approach for efficient and accurate dynamic simrank. *PVLDB*, *10*(9), 937–948. Retrieved from `http://www.vldb.org/pvldb/vol10/p937-jiang.pdf`

Karypis, G., & Kumar, V. (1998). A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, *20*(1), 359–392.

Kim, J., Choi, D.-H., & Li, C. (2019). Inves: Incremental partitioning-based verification for graph similarity search. In *Edbt* (pp. 229–240).

Kim, J. H. (2017). *Efficient node proximity and node significance computations in graphs* (Unpublished doctoral dissertation). Arizona State University.

Kim, J. H., Li, M.-L., Candan, K. S., & Sapino, M. L. (2017). Personalized pagerank in uncertain graphs with mutually exclusive edges. In *Proceedings of the 40th international acm sigir conference on research and development in information retrieval* (pp. 525–534).

Kinne, J., & Lenz, D. (2021). Predicting innovative firms using web mining and deep learning. *PloS one*, *16*(4), e0249071.

Kireyeu, V. (2021). Cluster dynamics studied with the phase-space minimum spanning tree approach. *Physical Review C*, *103*(5), 054905.

Kumar, R., Novak, J., & Tomkins, A. (2006). Structure and evolution of online social networks.

In *Kdd.*

Kusumoto, M., Maehara, T., & Kawarabayashi, K.-i. (2014). Scalable similarity search for simrank. In *Proceedings of the 2014 acm sigmod international conference on management of data* (pp. 325–336).

Laws, F., Michelbacher, L., Dorow, B., Scheible, C., Heid, U., & Schütze, H. (2010). A linguistically grounded graph model for bilingual lexicon extraction. In *Coling 2010: Posters* (pp. 614–622).

Lee, V. E. (2012). *Rolesim and rolematch: Role-based similarity and graph matching.* Kent State University.

Leskovec, J., Kleinberg, J. M., & Faloutsos, C. (2007). Graph evolution: Densification and shrinking diameters. *TKDD*, *1*(1), 2. Retrieved from `http://doi.acm.org/10.1145/1217299.1217301` doi: 10.1145/1217299.1217301

Li, C., Han, J., He, G., Jin, X., Sun, Y., Yu, Y., & Wu, T. (2010). Fast computation of simrank for static and dynamic information networks. In *Proceedings of the 13th international conference on extending database technology* (pp. 465–476).

Li, P., Liu, H., Yu, J. X., He, J., & Du, X. (2010). Fast single-pair SimRank computation. In *Sdm.*

Lin, Z., Lyu, M. R., & King, I. (2006). Pagesim: a novel link-based measure of web page aimilarity. In *Proceedings of the 15th international conference on world wide web* (pp. 1019–1020).

Lin, Z., Lyu, M. R., & King, I. (2012). Matchsim: a novel similarity measure based on maximum neighborhood matching. *Knowledge and information systems*, *32*(1), 141–166.

Linden, G. D., Jacobi, J. A., & Benson, E. A. (2001, July 24). *Collaborative recommendations using item-to-item similarity mappings.* Google Patents. (US Patent 6,266,649)

Liu, S., & Wang, S. (2016). Trajectory community discovery and recommendation by multi-source diffusion modeling. *IEEE Transactions on Knowledge and Data Engineering*, *29*(4), 898–911.

Lizorkin, D., Velikhov, P., Grinev, M., & Turdakov, D. (2008a). Accuracy estimate and optimization techniques for simrank computation. *Proceedings of the VLDB Endowment*, *1*(1), 422–433.

Lizorkin, D., Velikhov, P., Grinev, M. N., & Turdakov, D. (2008b). Accuracy estimate and optimization techniques for SimRank computation. *PVLDB*, *1*.

Lizorkin, D., Velikhov, P., Grinev, M. N., & Turdakov, D. (2010). Accuracy estimate and optimization techniques for simrank computation. *VLDB J.*, *19*(1), 45–66. Retrieved from `https://doi.org/10.1007/s00778-009-0168-8` doi: 10.1007/s00778-009-0168-8

Lloyd, S. (1982). Least squares quantization in pcm. *IEEE transactions on information theory*, *28*(2), 129–137.

Lofgren, P. A., Banerjee, S., Goel, A., & Seshadhri, C. (2014). Fast-ppr: Scaling personalized pagerank estimation for large graphs. In *Proceedings of the 20th acm sigkdd international conference on knowledge discovery and data mining* (pp. 1436–1445).

Ma, Y., Lin, H., & Lin, Y. (2011). Selecting related terms in query-logs using two-stage SimRank. In *Cikm.*

Machado, G. S., Bocek, T., Filitz, A., & Stiller, B. (2014). Measuring interactivity and geographical closeness of online social network users to support social recommendation systems. In *10th international conference on network and service management (cnsm) and workshop* (pp. 187–192).

Mehlhorn, K., & Sanders, P. (2008). *Algorithms and data structures: The basic toolbox.* Springer Science & Business Media.

Milo, T., Somech, A., & Youngmann, B. (2019). Boosting simrank with semantics. In *Proc. edbt* (pp. 1–12).

Muni Manasa, G., & Farooq, S. (n.d.). Fortified similarity integration in image-rich web sites using simlearn.

Nguyen, P., Tomeo, P., Di Noia, T., & Di Sciascio, E. (2015). An evaluation of simrank and personalized pagerank to build a recommender system for the web of data. In *Proceedings of the 24th international conference on world wide web* (pp. 1477–1482).

Ntoulas, A., Cho, J., & Olston, C. (2004). What's new on the web? the evolution of the web from a search engine perspective. In *Proceedings of the 13th international conference on world wide web* (pp. 1–12).

Potamias, M., Bonchi, F., Gionis, A., & Kollios, G. (2010). K-nearest neighbors in uncertain graphs. *Proceedings of the VLDB Endowment*, *3*(1-2), 997–1008.

Rao, P. N., Devi, T., Kaladhar, D., Sridhar, G., & Rao, A. A. (2009). A probabilistic neural network approach for protein superfamily classification. *Journal of Theoretical & Applied*

Information Technology, *6*(1).

Rothe, S., & Schütze, H. (2014). Cosimrank: A flexible & efficient graph-theoretic similarity measure. In *Proceedings of the 52nd annual meeting of the association for computational linguistics (volume 1: Long papers)* (pp. 1392–1402).

Rui, Y. (2013). *Urban growth modeling based on land-use changes and road network expansion* (Unpublished doctoral dissertation). KTH Royal Institute of Technology.

Sarlós, T., Benczúr, A. A., Csalogány, K., Fogaras, D., & Rácz, B. (2006). To randomize or not to randomize: space optimal summaries for hyperlink analysis. In *Proceedings of the 15th international conference on world wide web* (pp. 297–306).

Sedgewick, R., & Wayne, K. (2011). *Algorithms*. Addison-wesley professional.

Shahabi, C., Banaei-Kashani, F., Chen, Y.-S., & McLeod, D. (2001). Yoda: An accurate and scalable web-based recommendation system. In *International conference on cooperative information systems* (pp. 418–432).

Shao, Y., Cui, B., Chen, L., Liu, M., & Xie, X. (2015a). An efficient similarity search framework for simrank over large dynamic graphs. *Proceedings of the VLDB Endowment*, *8*(8), 838–849.

Shao, Y., Cui, B., Chen, L., Liu, M., & Xie, X. (2015b). An efficient similarity search framework for SimRank over large dynamic graphs. *PVLDB*, *8*(8), 838–849. Retrieved from `http://www.vldb.org/pvldb/vol8/p838-shao.pdf`

Shao, Y., Liu, J., Shi, S., Zhang, Y., & Cui, B. (2019). Fast de-anonymization of social networks with structural information. *Data Science and Engineering*, *4*(1), 76–92.

Smirnov, V., & Warnow, T. (2021). Magus: multiple sequence alignment using graph clustering. *Bioinformatics*, *37*(12), 1666–1672.

Sun, Y., Han, J., Yan, X., Yu, P. S., & Wu, T. (2011). PathSim: Meta path-based top-*k* similarity search in heterogeneous information networks. *PVLDB*, *4*(11), 992–1003. Retrieved from `http://www.vldb.org/pvldb/vol4/p992-sun.pdf`

Tamura, A., Watanabe, T., & Sumita, E. (2012). Bilingual lexicon extraction from comparable corpora using label propagation. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning* (pp. 24–36).

Tian, B., & Xiao, X. (2016). Sling: A near-optimal index structure for simrank. In *Proceedings* *of the 2016 international conference on management of data* (pp. 1859–1874).

Vegas Niño, O. T., Martínez Alzamora, F., & Tzatchkov, V. G. (2021). A decision support tool for water supply system decentralization via distribution network sectorization. *Processes*, *9*(4), 642.

Wang, Y., Wang, L., Li, Y., He, D., & Liu, T.-Y. (2013). A theoretical analysis of ndcg type ranking measures. In *Conference on learning theory* (pp. 25–54).

Wang, Y., Yu, S., Gu, Y., & Shun, J. (2021). Fast parallel algorithms for euclidean minimum spanning tree and hierarchical spatial clustering. In *Proceedings of the 2021 international conference on management of data* (pp. 1982–1995).

Wang, Z., Chen, Y., & Li, C. (2021). Opportunistic routing in mobile networks. *Opportunistic Networks: Fundamentals, Applications and Emerging Trends*, 243.

Wu, J. (2021). Construct a knowledge graph for china coronavirus (covid-19) patient information tracking. *Risk Management and Healthcare Policy*, *14*, 4321.

Wu, J., Wang, X., Feng, F., He, X., Chen, L., Lian, J., & Xie, X. (2021). Self-supervised graph learning for recommendation. In *Proceedings of the 44th international acm sigir conference on research and development in information retrieval* (pp. 726–735).

Wu, K., & Liu, X. (2021). Community detection in directed acyclic graphs of adversary interactions. *Physica A: Statistical Mechanics and its Applications*, *584*, 126370.

Xiangxue, W., Lunhui, X., & Kaixun, C. (2019). Data-driven short-term forecasting for urban road network traffic based on data processing and lstm-rnn. *Arabian Journal for Science and Engineering*, *44*(4), 3043–3060.

Yannakakis, M., & Gavril, F. (1980). Edge dominating sets in graphs. *SIAM Journal on Applied Mathematics*, *38*(3), 364–372.

Yu, W., Iranmanesh, S., Haldar, A., Zhang, M., & Ferhatosmanoglu, H. (2020). An axiomatic role similarity measure based on graph topology. In *Software foundations for data interoperability and large scale graph data analytics* (pp. 33–48). Springer.

Yu, W., & Lin, X. (2013). IRWR:Incremental random walk with restart. In *Sigir.*

Yu, W., Lin, X., & Zhang, W. (2013a). Towards efficient SimRank computation on large networks. In *Icde.*

Yu, W., Lin, X., & Zhang, W. (2013b). Towards efficient simrank computation on large networks. In *2013 ieee 29th international conference on data engineering (icde)* (pp. 601–612).

Yu, W., Lin, X., & Zhang, W. (2014a). Fast incremental simrank on link-evolving graphs. In *2014 ieee 30th international conference on data engineering* (pp. 304–315).

Yu, W., Lin, X., & Zhang, W. (2014b). Fast incremental simrank on link-evolving graphs. In *2014 ieee 30th international conference on data engineering* (pp. 304–315).

Yu, W., Lin, X., Zhang, W., Chang, L., & Pei, J. (2014). More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*.

Yu, W., Lin, X., Zhang, W., & McCann, J. A. (2015). Fast all-pairs simrank assessment on large graphs and bipartite domains. *IEEE Transactions on Knowledge and Data Engineering*, *27*(7), 1810–1823.

Yu, W., Lin, X., Zhang, W., & McCann, J. A. (2018a). Dynamical simrank search on time-varying networks. *The VLDB Journal*, *27*(1), 79–104.

Yu, W., Lin, X., Zhang, W., & McCann, J. A. (2018b). Dynamical SimRank search on time-varying networks. *VLDB J.*, *27*(1), 79–104. Retrieved from `https://doi.org/10.1007/s00778-017-0488-z` doi: 10.1007/s00778-017-0488-z

Yu, W., Lin, X., Zhang, W., Zhang, Y., & Le, J. (2012). SimFusion+: Extending SimFusion towards efficient estimation on large and dynamic networks. In *Sigir*.

Yu, W., & McCann, J. (2016). Random walk with restart over dynamic graphs. In *2016 ieee 16th international conference on data mining (icdm)* (pp. 589–598).

Yu, W., & McCann, J. A. (2014). Sig-sr: Simrank search over singular graphs. In *Proceedings of the 37th international acm sigir conference on research & development in information retrieval* (pp. 859–862).

Yu, W., & McCann, J. A. (2015a). Co-simmate: Quick retrieving all pairwise co-simrank scores..

Yu, W., & McCann, J. A. (2015b). Efficient partial-pairs SimRank search for large networks. *PVLDB*, *8*(5), 569–580. Retrieved from `http://www.vldb.org/pvldb/vol8/p569-yu.pdf`

Yu, W., & McCann, J. A. (2015c). High quality graph-based similarity search. In *Proceedings of the 38th international acm sigir conference on research and development in information retrieval* (pp. 83–92).

Yu, W., Zhang, W., Lin, X., Zhang, Q., & Le, J. (2012a). A space and time efficient algorithm for simrank computation. *World Wide Web*, *15*(3), 327–353.

Yu, W., Zhang, W., Lin, X., Zhang, Q., & Le, J. (2012b). A space and time efficient algorithm for SimRank computation. *World Wide Web*, *15*(3), 327–353. Retrieved from `https://doi.org/10.1007/s11280-010-0100-6` doi: 10.1007/s11280-010-0100-6

Zhang, M., Yang, L., Dong, Y., Wang, J., & Zhang, Q. (2021). Picture semantic similarity search based on bipartite network of picture-tag type. *PloS one*, *16*(11), e0259028.

Zhang, R., Rong, Y., Wu, Z., & Zhuo, Y. (2020). Trajectory similarity assessment on road networks via embedding learning. In *2020 ieee sixth international conference on multimedia big data (bigmm)* (pp. 1–8).

Zhao, P., Han, J., & Sun, Y. (2009). P-rank: a comprehensive structural similarity measure over information networks. In *Proceedings of the 18th acm conference on information and knowledge management* (pp. 553–562).

Zheng, W., Zou, L., Lian, X., Wang, D., & Zhao, D. (2014). Efficient graph similarity search over large graph databases. *IEEE Transactions on Knowledge and Data Engineering*, *27*(4), 964–978.

Appendices

A Configuration METIS

This appendix aims to illustrate the method to configure METIS algorithm in MATLAB. Considering to implement B_Lin algorithm for graph partition, we need to configure METIS algorithm (which is illustrated in Section A.1) firstly, we implement both algorithms in MATLAB 2017b. The basic information of machines and compliers is shown in Table A.1.

Table A.1: Basic Information of Configuration METIS

Machine and Compliers	Version
Computer System	Windows 7 (x64 bit)
C++ Complier	Visual Studio 2017 (x64 bit)
MATLAB	MATLAB 2017b
CMake	CMake (cmake-Gui) 3.10.0

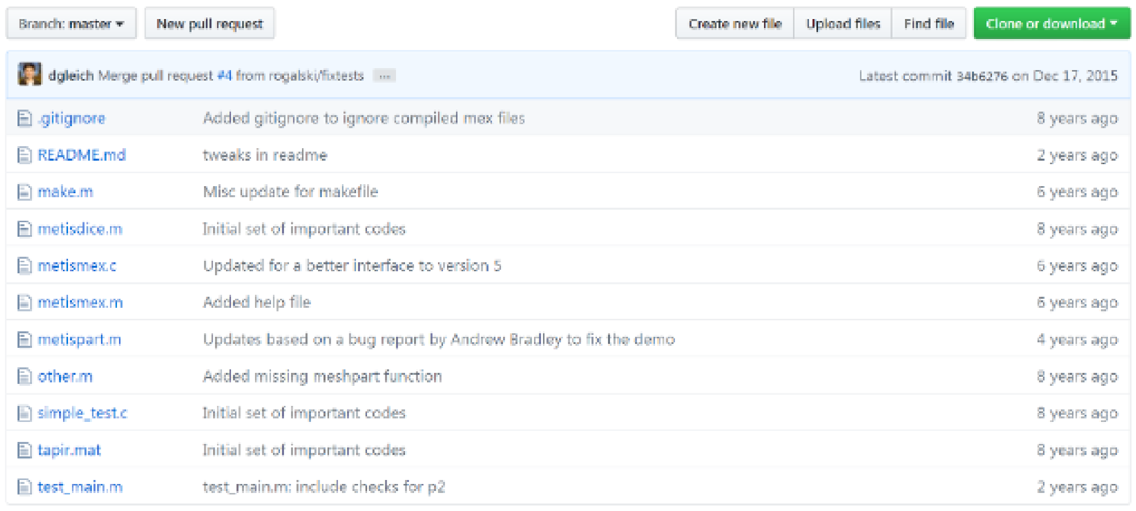
Note that the version of MATLAB, Visual Studio, CMake and computer system may affect the result of compilation. The basic steps to implement METIS in MATLAB on the system environment are shown in Appendix Table A.1.

A.1 Resource Acquisition

(1) Extract the Patched Version of METIS.

The METIS zip file is available to be downloaded from the following link:
https://github.com/helixhorned/metis/tree/pk.

There are installation instructions and algorithm C++ programs set in the METIS zip file, which is shown in Figure A.1. Since the installation instructions that come with the program package are not clear and detailed enough, thus this section introduces the problems and solutions encountered in using the program package in detail. Note that the location of the zip file decompression needs to be remembered, which will be used in the following steps.



dgleich Merge pull request #4 from rogalski/fixtests ... Latest commit 34b6276 on Dec 17, 2015

.gitignore	Added gitignore to ignore compiled mex files	8 years ago
README.md	tweaks in readme	2 years ago
make.m	Misc update for makefile	6 years ago
metisdice.m	Initial set of important codes	8 years ago
metismex.c	Updated for a better interface to version 5	6 years ago
metismex.m	Added help file	6 years ago
metispart.m	Updates based on a bug report by Andrew Bradley to fix the demo	4 years ago
other.m	Added missing meshpart function	8 years ago
simple_test.c	Initial set of important codes	8 years ago
tapir.mat	Initial set of important codes	8 years ago
test_main.m	test_main.m: include checks for p2	2 years ago

Figure A.1: Online Source of METIS

Extract the Metismex-master File.

Metismex-master File can be downloaded from the following link:
https://github.com/dgleich/metismex.
Metismex-master file has the compiling methods, and the MATLAB codes of METIS, the details of metismex-master file are shown in Figure A.2. The decompression location of metismex-master file should be the same as the METIS zip file.

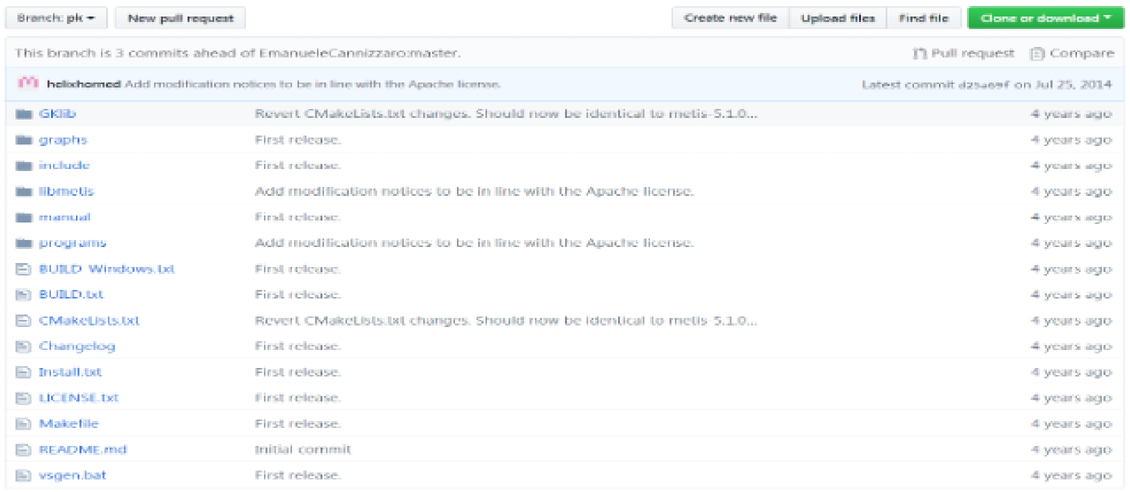
A.2 Compile METIS algorithm In MATLAB

Leveraging the METIS and metismex-master files, the next step is to compile METIS algorithm in MATLAB.

Generate Visual Studio project by CMake.

As previously stated, the METIS algorithm code set is compiled in C++; hence, in order to build it in MATLAB, the code set should be pre-processed in CMake.

First, enter the location of the METIS file as the location of the source code. Then create



Branch: pk ▾ | New pull request | Create new file | Upload files | Find file | Clone or download ▾

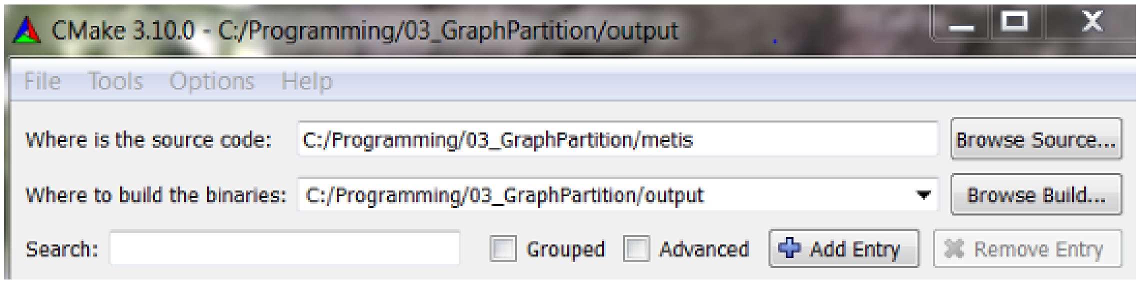
This branch is 3 commits ahead of EmanueleCannizzaro:master. Pull request Compare

helixhorned Add modification notices to be in line with the Apache license. Latest commit d25ae6f on Jul 25, 2014

GKlib	Revert CMakeLists.txt changes. Should now be identical to metis-5.1.0...	4 years ago
graphs	First release.	4 years ago
include	First release.	4 years ago
libmetis	Add modification notices to be in line with the Apache license.	4 years ago
manual	First release.	4 years ago
programs	Add modification notices to be in line with the Apache license.	4 years ago
BUILD_Windows.txt	First release.	4 years ago
BUILD.txt	First release.	4 years ago
CMakeLists.txt	Revert CMakeLists.txt changes. Should now be identical to metis-5.1.0...	4 years ago
Changelog	First release.	4 years ago
Install.txt	First release.	4 years ago
LICENSE.txt	First release.	4 years ago
Makefile	First release.	4 years ago
README.md	Initial commit	4 years ago
vsgen.bat	First release.	4 years ago

Figure A.2: The details of the metismex-master file

a new folder named "output" in the same location as METIS file, and use the location of the folder as the input location for the build binary file. An example of position input is shown in Figure A.3.



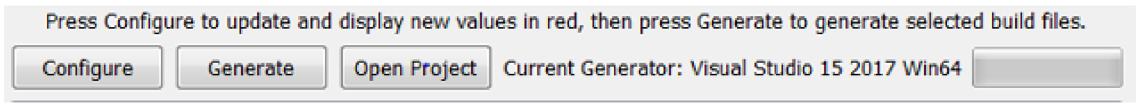
CMake 3.10.0 - C:/Programming/03_GraphPartition/output

Where is the source code: C:/Programming/03_GraphPartition/metis

Where to build the binaries: C:/Programming/03_GraphPartition/output

Figure A.3: Generate Visual Studio project by CMake (Phase 1)

Next, select "Visual Studio 15 2017 x64" to configure and generate the project. After the project is successfully generated, click "Open Project" to open the project, as shown in Figure A.4.



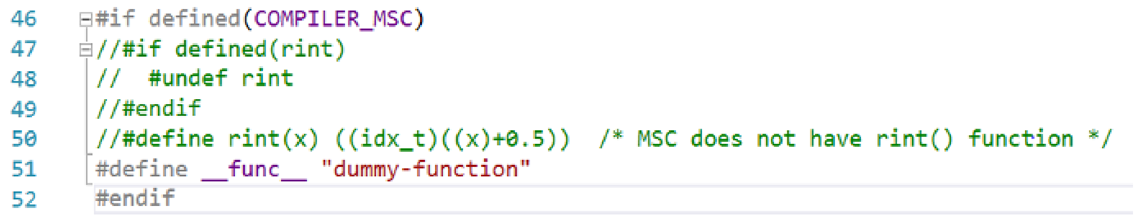
Press Configure to update and display new values in red, then press Generate to generate selected build files.

Current Generator: Visual Studio 15 2017 Win64

Figure A.4: Generate Visual Studio project by CMake (Phase 2)

Redefine clear.

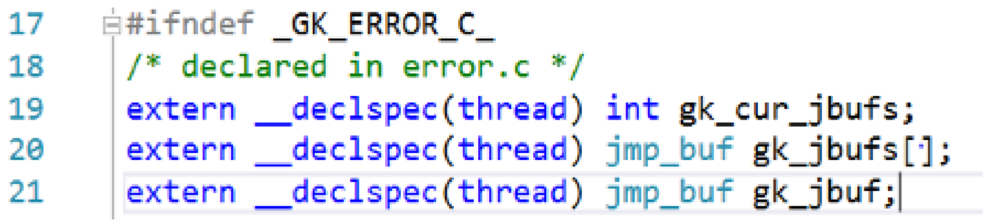
There are a few difficulties in Visual Studio that need to be handled before the library can be generated as a solution. The METIS code set includes three header files: metisbin.h, metislib.h, and gk arch.h, all redefine the 'rint' function for MSVC. Because METIS needs to be compiled in MATLAB, the definition of 'rint' function is not required. Thus, these 'rint' defining codes need to be removed from metisbin.h, metislib.h, and gk arch.h, as indicated in Figure A.5.



```
#if defined(COMPILER_MSC)
//#if defined(rint)
//  #undef rint
//#endif
//#define rint(x) ((idx_t)((x)+0.5))  /* MSC does not have rint() function */
#define __func__ "dummy-function"
#endif
```

Figure A.5: Redefinition in Several Header Files of METIS (Phase 1)

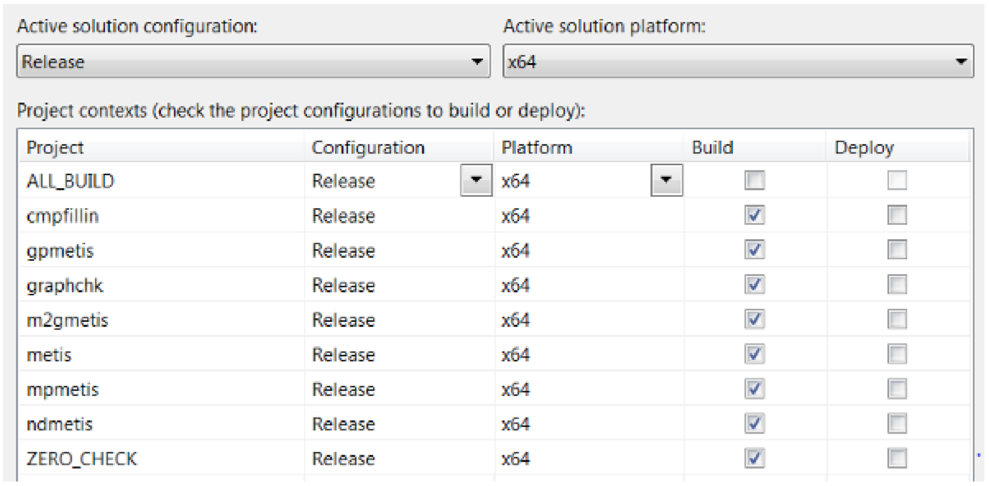
The next issue to address is the replacement of '_thread' with '_declspec(thread)' in the Klibgk externs.h file. Figure A.6 shows the details of the replacement.



```
#ifndef _GK_ERROR_C_
/* declared in error.c */
extern __declspec(thread) int gk_cur_jbufs;
extern __declspec(thread) jmp_buf gk_jbufs[];
extern __declspec(thread) jmp_buf gk_jbuf;
```

Figure A.6: Redefinition in Several Header Files of METIS (Phase 2)

The next step is to return to the METIS folder's location and open the METIS.sln file from the location in Visual Studio. In the "Release" mode, make sure to pick "x64" for the ALL BUILD target (by clicking "Build" and then "Configuration Manager"). Figure A.7 depicts the proper environment settings.



Active solution configuration: Release

Active solution platform: x64

Project contexts (check the project configurations to build or deploy):

Project	Configuration	Platform	Build	Deploy
ALL_BUILD	Release	x64	☐	☐
cmpfillin	Release	x64	☑	☐
gpmetis	Release	x64	☑	☐
graphchk	Release	x64	☑	☐
m2gmetis	Release	x64	☑	☐
metis	Release	x64	☑	☐
mpmetis	Release	x64	☑	☐
ndmetis	Release	x64	☑	☐
ZERO_CHECK	Release	x64	☑	☐

Figure A.7: Environment seeting

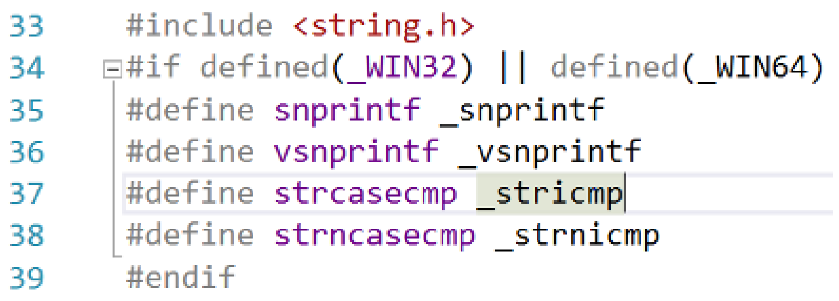
The project is ready to be compiled, and the last step is to save the solution metis.lib in the

Release folder.

Build MEX-function in MATLAB

Similar to the METIS file completed in Visual Studio, the metismex file has several issues that need to be addressed before complying.

Unzip the metixmex-master file into the METIS folder and rename metismex.c to metismex.cpp. Then, 'include<strings.h>' should be replaced with 'include<string.h>', which is after the line 'in-clude<string.h>'. The correct version of the metismex.cpp file is shown in Figure A.8.

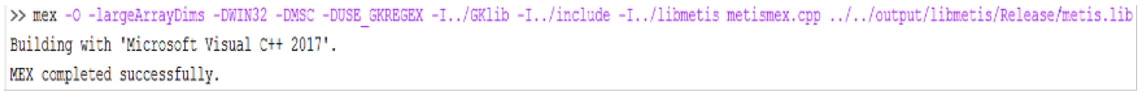


```
#include <string.h>
#if defined(_WIN32) || defined(_WIN64)
#define snprintf _snprintf
#define vsnprintf _vsnprintf
#define strcasecmp _stricmp
#define strncasecmp _strnicmp
#endif
```

Figure A.8: Build MEX-function in MATLAB

Then, go back to MATLAB and run the following command to compile the MEX file, and the correct result is shown in Figure A.9.

"mex -O -largeArrayDims -DWIN32 -DMSC -DUSE_GKREGEX -I../GKlib -../include -I../libmetis../ metismex.cpp(the directory location of metis.lib)"



```
>> mex -O -largeArrayDims -DWIN32 -DMSC -DUSE_GKREGEX -I../GKlib -I../include -I../libmetis metismex.cpp ../../output/libmetis/Release/metis.lib
Building with 'Microsoft Visual C++ 2017'.
MEX completed successfully.
```

Figure A.9: The code of MEX-file comply in MATLAB

A.3 Conclusion

The MATLAB METIS configuration has been accomplished. To summarise, we used CMake-Gui to produce the project and then addressed several issues in Visual Studio after downloading METIS.zip and metismex-master.zip. Then we unzip metis-master.zip into a subfolder of METIS and correct a few issues, such as redefining 'rint'. Finally, we construct the MEX function in MATLAB using the appropriate environment.

B SNAP Compilation

SNAP (Standard Network Analysis Platform) is a robust system for analysing and managing massive amounts of network data. SNAP's main code is built in C++, and it scales to enormous networks with millions of nodes and billions of edges with ease. SNAP's advantages include the ability to produce both random and regular graphs and the ability to calculate properties of big graphs and support attributes on nodes and edges. Another benefit of SNAP is that the node and edge attribution can change dynamically during the computation. Note that the application and system vision may affect SNAP compilation. The basic information of machines and compliers is shown in Table B.1.

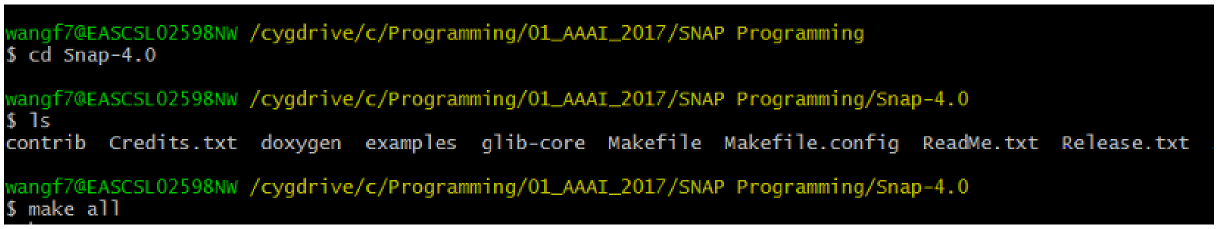
The steps of SNAP compilation are illustrated as follows.

Resource Acquisition.

The lasted vision (Jul 27, 2017) of SNAP can be downloaded from the following link:
https://snap.stanford.edu/snap/download.html
and the contents of the SNAP resource package are shown in Table B.2:

Comply SNAP with Cygwin.

Install the Cygwin package with the C++ compiler GCC package before making SNAP compatible with Cygwin. Enter the Snap-4.0 directory of Cygwin, and then run " make all" in this directory to compile the core SNAP library and all application examples. The specific process method is shown in Figure B.1.



```
wangf7@EASCSL02598NW /cygdrive/c/Programming/01_AAAI_2017/SNAP Programming
$ cd Snap-4.0

wangf7@EASCSL02598NW /cygdrive/c/Programming/01_AAAI_2017/SNAP Programming/Snap-4.0
$ ls
contrib  Credits.txt  doxygen  examples  glib-core  Makefile  Makefile.config  ReadMe.txt  Release.txt

wangf7@EASCSL02598NW /cygdrive/c/Programming/01_AAAI_2017/SNAP Programming/Snap-4.0
$ make all
```

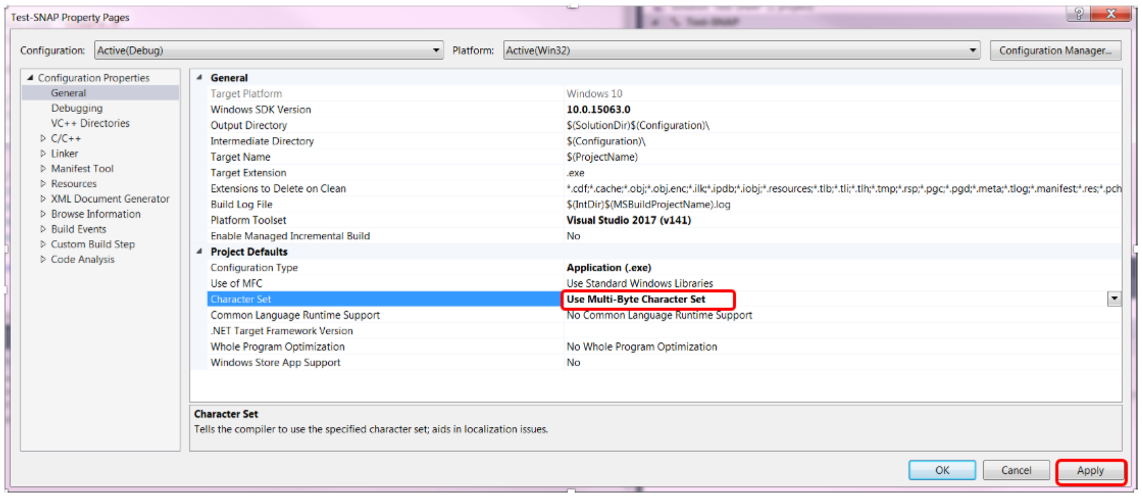
Figure B.1: Comply SNAP with Cygwin

Machine and Compliers	Version
Computer System	Windows 7 (x64 bit)
C++ Complier	Visual Studio 2017 (x64 bit)
MATLAB	MATLAB 2017b
CMake	CMake (cmake-Gui) 3.10.0
Cygwin	Cygwin64 Terminal

Table B.1: Basic Information of SNAP Compilation

Build a New Visual Studio Project.

First, build a new C++ project and configure SNAP in Visual Studio. Then, to change the character set to Multi-byte in this stage. Moving mouse hover your project and right-click on it. Go to Properties –> Configuration Properties –> General –> Projects Defaults –> Character Set –> Select "Use Multi-Byte Character Set". The setting window is shown in Figure B.2.



General	
Target Platform	Windows 10
Windows SDK Version	**10.0.15063.0**
Output Directory	$(SolutionDir)$(Configuration)\
Intermediate Directory	$(Configuration)\
Target Name	$(ProjectName)
Target Extension	.exe
Extensions to Delete on Clean	*.cdf;*.cache;*.obj;*.obj.enc;*.ilk;*.ipdb;*.iobj;*.resources;*.tlb;*.tli;*.tlh;*.tmp;*.rsp;*.pgc;*.pgd;*.meta;*.tlog;*.manifest;*.res;*.pch
Build Log File	$(IntDir)$(MSBuildProjectName).log
Platform Toolset	**Visual Studio 2017 (v141)**
Enable Managed Incremental Build	No
Project Defaults	
Configuration Type	**Application (.exe)**
Use of MFC	Use Standard Windows Libraries
Character Set	**Use Multi-Byte Character Set**
Common Language Runtime Support	No Common Language Runtime Support
.NET Target Framework Version	
Whole Program Optimization	No Whole Program Optimization
Windows Store App Support	No

Character Set
Tells the compiler to use the specified character set; aids in localization issues.

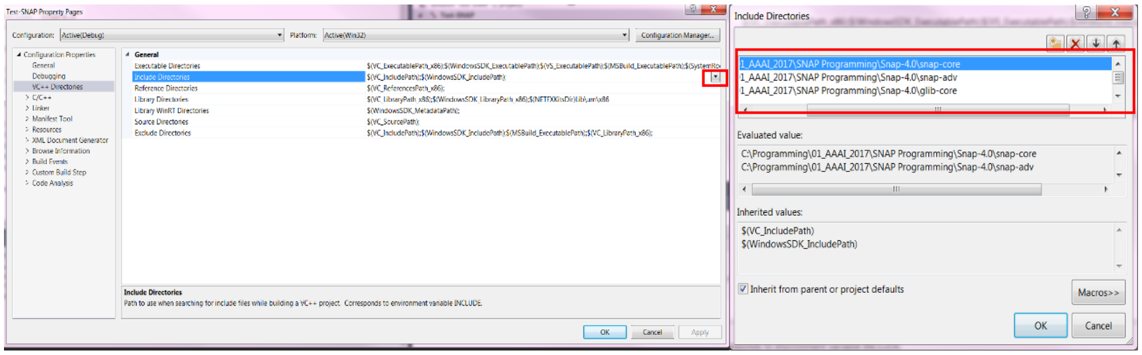
Figure B.2: Configure SNAP in Visual Studio

Next is to add the location of SNAP into VC++ directories. Moving mouse hover your

Component	Content
snap-core	The core SNAP graph library
snap-adv	Advanced SNAP components, which is not in the core but used by examples
snap-exp	experimental SNAP components (in develepment)
examples	small sample applications that demonstrate SNAP functionality
tutorials	demonstrating use of various classes
glib-core	STL-like library that implements basic data structures, *e.g.,* vectors(TVec), hash-tables(THash), strings(TStr), provides serialization etc.
test	unit tests for various classes
doxygen	SNAP reference manuals

Table B.2: Code set of SNAP Compilation

project and right-click on it. Go to Properties –> Configuration Properties –> VC++ Directories –> Include Directories. The environment setting window is shown in Figure B.3.



Test-SNAP Property Pages

Configuration: Active(Debug) Platform: Active(Win32)

General	
Executable Directories	$(VC_ExecutablePath_x86);$(WindowsSDK_ExecutablePath);$(VS_ExecutablePath);$(MSBuild_ExecutablePath);$(SystemRo
Include Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);
Reference Directories	$(VC_ReferencesPath_x86);
Library Directories	$(VC_LibraryPath_x86);$(WindowsSDK_LibraryPath_x86);$(NETFXKitsDir)Lib\um\x86
Library WinRT Directories	$(WindowsSDK_MetadataPath);
Source Directories	$(VC_SourcePath);
Exclude Directories	$(VC_IncludePath);$(WindowsSDK_IncludePath);$(MSBuild_ExecutablePath);$(VC_LibraryPath_x86);

Include Directories
Path to use when searching for include files while building a VC++ project. Corresponds to environment variable INCLUDE.

Include Directories

```
1_AAAI_2017\SNAP Programming\Snap-4.0\snap-core
1_AAAI_2017\SNAP Programming\Snap-4.0\snap-adv
1_AAAI_2017\SNAP Programming\Snap-4.0\glib-core
```

Evaluated value:

```
C:\Programming\01_AAAI_2017\SNAP Programming\Snap-4.0\snap-core
C:\Programming\01_AAAI_2017\SNAP Programming\Snap-4.0\snap-adv
```

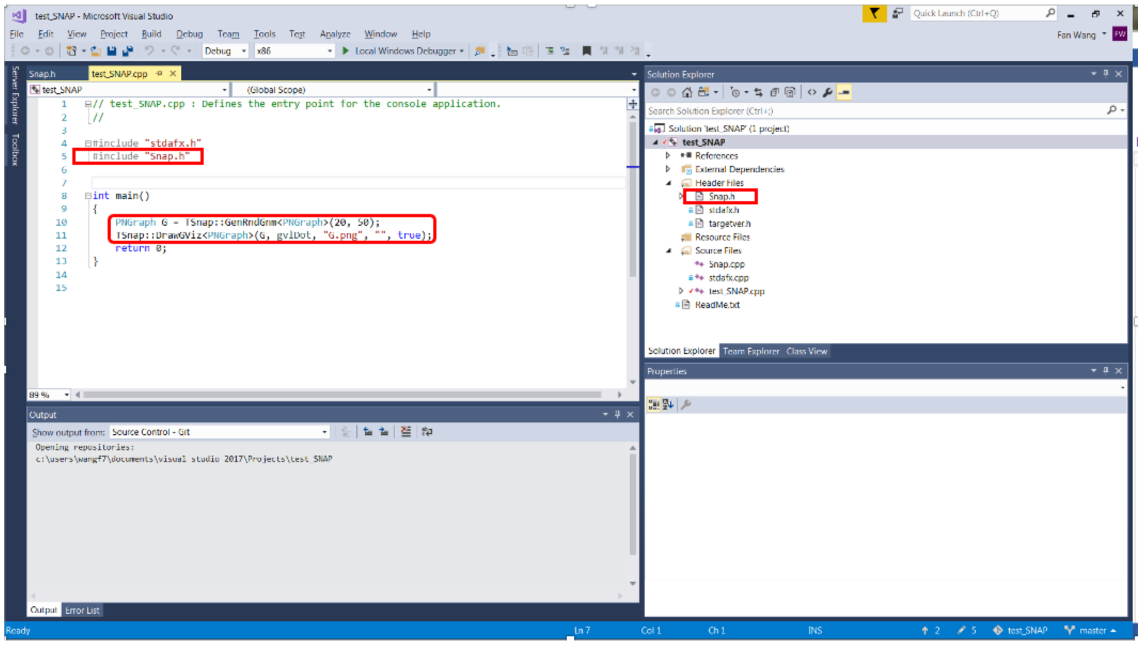
Inherited values:

```
$(VC_IncludePath)
$(WindowsSDK_IncludePath)
```

[✓] Inherit from parent or project defaults

Figure B.3: Add the location of SNAP into VC++ directories

Finally, add Snap.h into head file list of the project, and write #include "Snap.h" in test_SNAP.main. Input code to call SNAP function. The result is shown in Figure B.4.



test_SNAP - Microsoft Visual Studio

test_SNAP.cpp

```
// test_SNAP.cpp : Defines the entry point for the console application.
//

#include "stdafx.h"
#include "Snap.h"


int main()
{
    PNGraph G = TSnap::GenRndGnm<PNGraph>(20, 50);
    TSnap::DrawGViz<PNGraph>(G, gvlDot, "G.png", "", true);
    return 0;
}
```

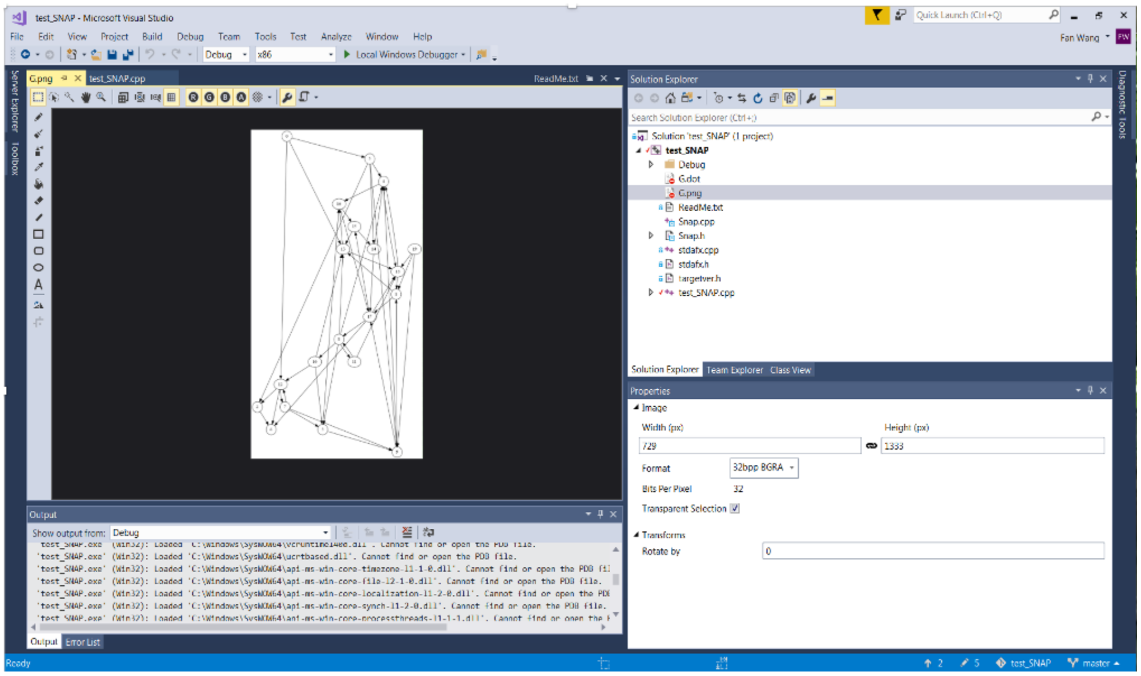
Output

Show output from: Source Control - Git

```
Opening repositories:
c:\users\wangf7\documents\visual studio 2017\Projects\test_SNAP
```

Figure B.4: Add SNAP head file into project

SNAP compilation has been done, and the result of example is shown in Figure B.5.



Properties

Image

Width (px): 729
Height (px): 1333
Format: 32bpp BGRA
Bits Per Pixel: 32
Transparent Selection ✓

Transforms

Rotate by: 0

Output

Show output from: Debug

```
'test_SNAP.exe' (Win32): Loaded 'C:\Windows\SysWOW64\vcruntime140d.dll'. Cannot find or open the PDB file.
'test_SNAP.exe' (Win32): Loaded 'C:\Windows\SysWOW64\ucrtbased.dll'. Cannot find or open the PDB file.
'test_SNAP.exe' (Win32): Loaded 'C:\Windows\SysWOW64\api-ms-win-core-timezone-l1-1-0.dll'. Cannot find or open the PDB fil
'test_SNAP.exe' (Win32): Loaded 'C:\Windows\SysWOW64\api-ms-win-core-file-l2-1-0.dll'. Cannot find or open the PDB file.
'test_SNAP.exe' (Win32): Loaded 'C:\Windows\SysWOW64\api-ms-win-core-localization-l1-2-0.dll'. Cannot find or open the PDB
'test_SNAP.exe' (Win32): Loaded 'C:\Windows\SysWOW64\api-ms-win-core-synch-l1-2-0.dll'. Cannot find or open the PDB file.
'test_SNAP.exe' (Win32): Loaded 'C:\Windows\SysWOW64\api-ms-win-core-processthreads-l1-1-1.dll'. Cannot find or open the P
```

Figure B.5: The Result of SNAP Example File