Efficient Node-to-Node Relevance Assessment Based on Hyperlinks

by

Weiren Yu

THE UNIVERSITY OF
NEW SOUTH WALES

SYDNEY • AUSTRALIA

A THESIS SUBMITTED IN FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
IN THE SCHOOL
OF
COMPUTER SCIENCE AND ENGINEERING
(January 2014)

Supervisor: Prof. Xuemin Lin

PLEASE TYPE

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: Yu

First name: Weiren Other name/s: N/A

Abbreviation for degree as given in the University calendar: PhD

School: School of Computer Science and Engineering Faculty: Faculty of Engineering

Title: Efficient Node-to-Node Relevance Assessment Based on Hyperlinks

Abstract 350 words maximum: (PLEASE TYPE)

Many ubiquitous applications need to assess relevance between two objects based on hyperlink structure. Typical examples include web page ranking, co-citation analysis, collaborative filtering, outlier detection, graph clustering, and nearest neighbor search. These applications have spurred growing interest in a powerful class of relevance assessment, known as link analysis. Link-based relevance assessment aims to assign a similarity score to each pair of objects based purely on the structure of a network, in contrast to the conventional text-based counterpart that heavily hinges on the content of objects.

In reality, networks are often large and frequently evolve with small changes over time. Due to the large scale and dynamic nature of the Internet, a fundamental challenge in link analysis is to design a satisfactory general-purpose similarity measure, which not only can well simulate human judgment behavior, but also has desirable computational efficiency, together with a succinct and elegant representation. To address the challenge, this thesis focuses on effective link-based relevance assessment over large and dynamic networks, which encompasses (1) computational efficiency on large networks, (2) incremental update on dynamic networks, and (3) semantics improvement of existing similarity measures.

More specifically, our contributions are summarized as follows:

(1) We propose efficient techniques for assessing SimRank relevance on large networks and bipartite domains.
(2) We design a novel paradigm for incrementally assessing SimRank on link-evolving networks.
(3) We provide efficient techniques for Penetrating-Rank (P-Rank) assessment on large networks.
(4) We study the incremental assessment of Random Walk with Restart (RWR) proximities in dynamic networks.
(5) We extend SimFusion model towards efficient relevance assessment on large and dynamic networks.
(6) We present a novel link-based model, SimRank*, for improving the semantic richness of SimRank and RWR.

We conduct comprehensive experiments on both real and synthetic datasets to demonstrate the superiority of our techniques against the state-of-the-art competitors.

Declaration relating to disposition of project thesis/dissertation

I hereby grant to the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or in part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all property rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstracts International (this is applicable to doctoral theses only).

Weiren Yu Chengyuan Zhang Feb. 2014

Signature Witness Date

The University recognises that there may be exceptional circumstances requiring restrictions on copying or conditions on use. Requests for restriction for a period of up to 2 years must be made in writing. Requests for a longer period of restriction may be considered in exceptional circumstances and require the approval of the Dean of Graduate Research.

FOR OFFICE USE ONLY Date of completion of requirements for Award:

THIS SHEET IS TO BE GLUED TO THE INSIDE FRONT COVER OF THE THESIS

Originality Statement

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.

Name: Weiren Yu

Signed ..Weiren Yu....

Date .....Feb., 2014....

Abstract

Many ubiquitous applications need to assess relevance between two objects based on hyperlink structure. Typical examples include web page ranking, co-citation analysis, collaborative filtering, outlier detection, graph clustering, and nearest neighbor search. These applications have spurred growing interest in a powerful class of relevance assessment, known as link analysis. Link-based relevance assessment aims to assign a similarity score to each pair of objects based purely on the structure of a network, in contrast to the conventional text-based counterpart that heavily hinges on the content of objects.

In reality, networks are often large and frequently evolve with small changes over time. Due to the large scale and dynamic nature of the Internet, a fundamental challenge in link analysis is to design a satisfactory general-purpose similarity measure, which not only can well simulate human judgment behavior, but also has desirable computational efficiency, together with a succinct and elegant representation. To address the challenge, this thesis focuses on effective link-based relevance assessment over large and dynamic networks, which encompasses (1) computational efficiency on large networks, (2) incremental update on dynamic networks, and (3) semantics improvement of existing similarity measures.

More specifically, our contributions are summarized as follows:

(1) We propose efficient techniques for assessing SimRank relevance on large networks and bipartite domains. First, we exploit a novel clustering strategy for eliminating duplicate computations occurring in partial sums to accelerate SimRank for each iteration. Then, we introduce a new differential SimRank equation to reduce the total number of SimRank iterations. Thirdly, in bipartite domains, we also speed up the computation of the Minimax SimRank variation via edge concentration.

(2) We design a novel paradigm for incrementally assessing SimRank on link-evolving networks. Unlike the prior method maintaining updates to singular value decomposition, we first characterize the SimRank update matrix, in response to every link update, as a rank-one Sylvester equation. We then leverage an effective pruning technique capturing

the "affected areas" of the SimRank update matrix to skip unnecessary computations.

(3) We provide efficient techniques for Penetrating-Rank (P-Rank) assessment on large networks. First, we estimate the accuracy for P-Rank iterations. Then, we analyze the stability of P-Rank by obtaining a tight bound on its condition number. Finally, we propose efficient algorithms for P-Rank assessment on digraphs and undirected networks.

(4) We investigate the incremental assessment of Random Walk with Restart (RWR) proximities in dynamic networks. The prior attempt of RWR deploys k-dash to find top-k highest proximity nodes for a given query, involving an approximate strategy to incrementally estimate upper proximity bounds. In contrast, we propose a fast incremental paradigm for assessing RWR via linear combinations of vectors without loss of exactness.

(5) We extend SimFusion model towards efficient relevance assessment on large and dynamic networks. As opposed to the original SimFusion that utilizes a Unified Relationship Matrix (URM) to represent latent relationships among heterogeneous data, we present SimFusion+ based on a notion of the Unified Adjacency Matrix (UAM), to resolve the trivial solution and the divergence issues of SimFusion. We also develop fast algorithms to speed up the assessment of SimFusion+ on large and dynamical networks.

(6) We present a novel link-based model, SimRank*, for improving the semantic richness of SimRank and RWR. First, we justify that SimRank* can resolve an undesirable "zero-similarity" property in SimRank and RWR. Then, we propose a closed form of SimRank*, to enrich relevance semantics without suffering from increased computational cost. Finally, we devise a heuristic to speed up SimRank* assessment on large networks.

We conduct comprehensive experiments on both real and synthetic datasets to demonstrate the superiority of our techniques against the state-of-the-art competitors.

Acknowledgements

First and foremost, I am particularly indebted to my supervisor Prof. Xuemin Lin. He has provided me with immense help and excellent guidance throughout my PhD. He not only steered me in the correct direction during the problem definition and solution stages, but also provided invaluable help in disseminating my work to the research community. My presentation and writing skills will always bear the mark of his guidance.

I offer my heartfelt thanks to Dr. Wenjie Zhang for her suggestions that led to several improvements. Her multiple readings of the manuscript and corrections were invaluable. I also thank Prof. Raymond Wong for giving me the opportunity to intern at National ICT Australia (NICTA) on a very interesting and impressive research project, which was a great learning experience for me, and molded the research direction that I took during the rest of my PhD. I am thankful to Prof. Jian Pei for his insightful suggestions and conversations when I was an intern at East Normal China University. I am also grateful to Dr. Ying Zhang and Dr. Lijun Chang for providing a most fruitful collaboration.

I thank all the members of the UNSW Database Group for their useful comments and discussions. Special thanks goes to Dr. Muhammad Aamir Cheema for organizing our group meeting smoothly. I also thank Prof. William Wilson and Dr. Geoff Whale, two exceptionally fine teachers, for giving me a chance to practise my tutoring skills. During my stay at UNSW, I have enjoyed the company of many close friends. Our interactions have ranged from fun activities to serious technical discussions. I thank, to name a few, Haichuan Shang, Chuan Xiao, Gaoping Zhu, Ke Zhu, Zhitao Shen, Xiang Zhao, Liming Zhan, Pengjie Ye, Yang Wang, Lin Wu, Chengyuan Zhang, Xiaoyang Wang, Shiyu Yang, Longbin Lai, Xing Feng, Xiang Wang, Long Yuan, Shenglu Wang, Jianye Yang, for this wonderful time.

Finally, neither this thesis nor anything else I have done in my career would have been possible without the love, encouragement, and unwavering support from my parents and my girl friend, Qian. I dedicate this thesis to all of them.

Publications Involved in Thesis

1. W. Yu, X. Lin, and W. Zhang. Towards Efficient SimRank Computation on Large Networks. *The 29th IEEE International Conference on Data Engineering* (**ICDE 2013**), Brisbane, Australia, 2013.

2. W. Yu, X. Lin, and W. Zhang. Fast All-Pairs SimRank Assessment on Large Graphs and Bipartite Domains. *To appear in IEEE Transactions on Knowledge and Data Engineering* (**TKDE**), 2014.

3. W. Yu, X. Lin, and W. Zhang. Fast Incremental SimRank on Link-Evolving Graphs. *The 30th IEEE International Conference on Data Engineering* (**ICDE 2014**), Chicago, USA, 2014.

4. W. Yu, J. Le, X. Lin, and W. Zhang. On the Efficiency of Estimating Penetrating Rank on Large Graphs. *The 24th International Conference on Scientific and Statistical DB Management* (**SSDBM 2012**), Crete, Greece, 2012.

5. W. Yu, and X. Lin. IRWR: Incremental Random Walk with Restart. *The 36th ACM SIGIR International Conference* (**SIGIR 2013**), Dublin, Ireland, 2013.

6. W. Yu, X. Lin, W. Zhang, Y. Zhang, and J. Le. SimFusion+: Extending SimFusion towards Efficient Estimation on Large and Dynamic Networks. *The 35th ACM SIGIR International Conference* (**SIGIR 2012**), Portland, USA, 2012.

7. W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei. More is Simpler: Effectively and Efficiently Assessing Node-Pair Similarities Based on Hyperlinks. *The 39th International Conference on Very Large Data Base* (**PVLDB 2013**), 2013.

Below lists the relationship between the publications and the thesis chapters.

Chapters	**Pub No.**	**Problems Studied**
2	1,2	Fast SimRank on Large Networks and Bipartite Domains
3	3	Incremental SimRank on Link-Evolving Networks
4	4	Fast Penetrating-Rank Search on Large Networks
5	5	Incremental RWR Proximity on Dynamic Networks
6	6	Efficient SimFusion+ on Large and Dynamic Networks
7	7	A Novel Effective Model for Node-Pair Similarity Assessment

Contents

List of Figures

Chapter 1

Introduction

The advent and increasing importance of many proliferative application areas — link analysis, structural information search, recommender systems, and graph databases — have led to a growing need to assess node-to-node relevance based on link structure. Existing techniques for link-based relevance assessment do not support large and dynamic network data. Thus, applications are left to either: (1) Limit the similarity computation in small networks with no more than thousands of nodes; or (2) Reassess all pairs of relevance from scratch when a network is frequently updated with changes. While the former method imposes a significant restriction on many large-scale real applications, the latter results in considerable amounts of duplicate recomputational efforts and prohibitively high time complexity to deal with real evolving networks.

This thesis develops innovative techniques for efficient node-to-node relevance assessment on large and dynamic network data. Such techniques mainly involve (1) taming the computational complexity for similarity assessment over large networks, (2) supporting incremental updates when a network is constantly updated, as well as (3) enriching the semantics of relevance scoring functions in a systematic fashion. Section 1.1 highlights the importance of link-based relevance assessment that arises in modern-day applications. Section 1.2 describes fundamentals of relevance assessment used in the rest of the thesis. Section 1.3 introduces technical challenges involved in efficient relevance assessment on

large and dynamic networks, and outlines the main contributions made by this thesis. Section 1.4 provides a broad overview of related work in the general area of relevance assessment. (More detailed comparisons between prior work and specific contributions of this thesis will appear in later chapters.)

1.1 Real Applications

Below are several examples of relevance assessment arising in real-world applications, motivating the need for efficient techniques to deal with such assessment on large and dynamic network data.

- **Recommender Systems.** Recommender systems are based on analyzing a large amount of information on users' preferences and providing personalized recommendations of items to a user. A recent recommendation technique in [KSJ09] is based on relevance assessment on a graph that links users to tags and tags to items. Some additional information, *e.g.,* friendship and social tagging embedded in social knowledge, is also incorporated to improve the accuracy of item recommendations.

- **Bibliometrics.** Bibliometrics studies often require a relevance assessment for measuring documents based on citation relationship. The methods of co-citation [Sma73] and bibliographic coupling [Kes63a] are two most noteworthy metrics. Both measures, however, only use the information of common immediate neighbors to assess relevance between two documents. Recently, co-citation and bibliographic coupling have been generalized by using the entire graph structure to assess the relevance between documents [JW02].

- **Automated Image Annotation.** Automated image annotation aims to automatically assign caption keywords to a query image. A graph-based automatic captioning method was introduced by [PYFD04], where images and caption keywords are regarded as nodes in a mixed media graph. Then, link-based relevance

assessment can be applied to measure the correlations between the query image and the caption keywords.

- **Graph Clustering.** Graph clustering aims to partition nodes in a network into several different densely connected components based on node connectivity and/or neighborhood similarity. Many graph clustering methods focus on the topological structure of a graph partition with the aim to achieve a cohesive internal structure. The link-based relevance assessment between two nodes can be regarded as the node distance in a graph. One recent work in [SHZ+09] proposes RankClus to integrate relevance assessment with clustering in large network analysis.

- **Social Networks.** A fundamental task in social networks is to answer the question "which new interactions among social network members are more likely to occur in the near future?" A recent work in [LNK03] has investigated this question by assessing the relevance between two members. A key observation underlying this approach is that the topological structure of the social network may suggest many new collaborations. For instance, two members who are close in the network will have many friends in common, and thus are more likely to collaborate in the future.

1.2 Fundamentals of Relevance Assessment

We revisit and define the basic formal concepts needed in the remainder of the thesis.

Definition 1.1. *A directed graph* (or digraph) is an ordered pair $G = (V, E)$ with

- V – a set whose elements are called *nodes* or *vertices*, and
- E – a set of ordered pairs of nodes, called *directed edges*.

$|V|$ and $|E|$ denote *the number (cardinality) of nodes and edges*, respectively, in G. □

Definition 1.2. Given a graph $G = (V, E)$, for any node $a \in V$, we define

- $\mathcal{I}(a)$ – the *in-neighbor set* of node a, *i.e.,* all nodes that have a link to a :

$$\mathcal{I}(a) = \{u \in V | (u, a) \in E\}.$$

- $\mathcal{O}(a)$ – the *out-neighbor set* of node a, *i.e.,* all nodes that node v has a link to :

$$\mathcal{O}(a) = \{w \in V | (a, w) \in E\}.$$

- $|\mathcal{I}(a)|$ – the *in-degree* of node a, *i.e.,* the cardinality of $\mathcal{I}(a)$.

- $|\mathcal{O}(a)|$ – the *out-degree* of node a, *i.e.,* the cardinality of $\mathcal{O}(a)$. □

Using the aforementioned notations, the famous Google PageRank [Ber05] can be formulated as follows.

Definition 1.3 (PageRank)**.** Given a graph $G = (V, E)$, the PageRank value for any page u is defined as:

$$Pr(u) = C \cdot \sum_{v \in \mathcal{I}(u)} \frac{Pr(v)}{|\mathcal{O}(v)|} + (1 - C) \cdot \frac{1}{|V|},$$

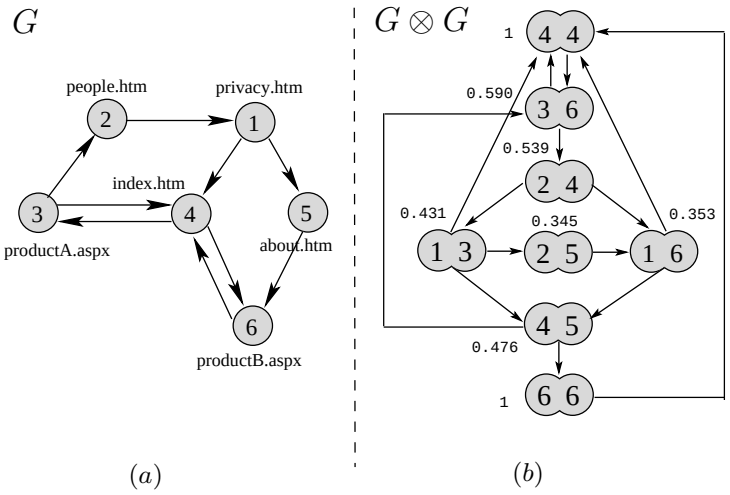
where $C \in (0, 1)$ is a constant decay factor (or a damping factor). Empirically, C is usually set to 0.85. □

PageRank was proposed by Larry Page [Ber05] to rank web pages based on hyperlinks in the search engine results. It is an effective approach of measuring the importance of web pages (nodes), by assigning a relevance score to each node.

In contrast to PageRank which is *query-independent*, Random walk with restart (RWR) [TFP06] has emerged recently as an appealing ranking algorithm relying on user queries. The formulation of RWR is a slight modification of PageRank as follows.

Definition 1.4 (Random Walk with Restart)**.** Given a graph $G = (V, E)$, the RWR proximity for any page u with respect to query q is defined as:

$$P_q(u) = C \cdot \sum_{v \in \mathcal{I}(u)} \frac{P_q(v)}{|\mathcal{O}(v)|} + (1 - C) \cdot \begin{cases} 0, & q \neq u; \\ 1, & q = u. \end{cases}$$ □



(a) (b)

Figure 1.1: SimRank on each pair of G *vs.* SimRank on each node in $G \otimes G$

Definition 1.5. *A tensor product graph* $G \otimes G = (V \otimes V, E \otimes E)$ of a graph G with itself is a graph such that

- $\forall (a, b) \in V \otimes V$ if $a, b \in V$;
- $\forall ((a_1, b_1), (a_2, b_2)) \in E \otimes E$ if $(a_1, a_2) \in E$, and $(b_1, b_2) \in E$. □

Example 1.6. Figure 1.1 depicts a digraph G (with nodes indexed by integers, denoting web pages, and edges hyperlinks) and its induced tensor graph $G \otimes G$. It can be seen that, in $G \otimes G$, edge $((3, 6), (2, 4))$ corresponds to two edges $(3, 2)$ and $(6, 4)$ in G. □

Based on the definition of the tensor product graph, we next formulate SimRank similarity, which was introduced by Jeh and Widom in [JW02].

Definition 1.7 (SimRank)**.** Given a graph $G = (V, E)$, let $s : V \otimes V \to [0, 1] \subset \mathbb{R}$ be a real-valued scoring function on $G \otimes G$ defined as

$$s(a, b) = \begin{cases} 1, & a = b; \\ \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} s(i, j), & \mathcal{I}(a), \mathcal{I}(b) \neq \varnothing; \\ 0, & \text{otherwise.} \end{cases} \tag{1.1}$$

where $C \in (0, 1)$ is a constant decay factor (or a damping factor). We call $s(a, b)$ the *SimRank similarity score between nodes* a *and* b. □

Figure 1.1 illustrates that SimRank propagates similarities from pair to pair in G associated with the propagation from node to node in $G \otimes G$ with a decay factor $C = 0.8$

Definition 1.8. *A bipartite graph (or bigraph)* is a graph $B = (U \cup V, E)$, whose nodes can be divided into two disjoint sets U and V, such that every edge in E connects a node in U to one in V. □

A bipartite version of SimRank, called Minimax SimRank [JW02], is defined below.

Definition 1.9 (Minimax SimRank variation)**.** Given a bipartite graph $B = (U \cup V, E)$, for every two distinct vertices A and B in V, the similarity of the Minimax SimRank variation, denoted as $s(A, B)$, is defined as follows:

$$\begin{aligned} s^A(A,B) &= \frac{C}{|\mathcal{O}(A)|} \sum_{i \in \mathcal{O}(A)} \max_{j \in \mathcal{O}(B)} s(i,j), \\ s^B(A,B) &= \frac{C}{|\mathcal{O}(B)|} \sum_{j \in \mathcal{O}(B)} \max_{i \in \mathcal{O}(A)} s(i,j), \\ s(A,B) &= \min\{s^A(A,B), s^B(A,B)\}. \quad \square \end{aligned}$$

To make the thesis self-contained, we also revisit some mathematical definitions that will be useful throughout the thesis.

We shall use the bold symbol $\mathbf{X} = (x_{i,j}) \in \mathbb{R}^{n \times m}$ to denote a matrix of size $n \times m$. Based on this, we define the following notations:

- $[\mathbf{X}]_{i,j}$ (or $x_{i,j}$) – the (i,j)-entry of matrix $\mathbf{X}$;
- $[\mathbf{X}]_{i,\star}$ – the i-th row of matrix $\mathbf{X}$;
- $[\mathbf{X}]_{\star,j}$ – the j-th column of matrix $\mathbf{X}$;
- $\mathbf{X}^T$ – the transpose of matrix $\mathbf{X}$, *i.e.*, $[\mathbf{X}]_{i,j} = [\mathbf{X}^T]_{j,i}$.

Definition 1.10 (tensor product)**.** The *tensor product (or Kroneckor product)* of two matrices $\mathbf{X} \in \mathbb{R}^{p \times q}$ and $\mathbf{Y} \in \mathbb{R}^{r \times s}$ is the $pr \times qs$ matrix

$$\mathbf{X} \otimes \mathbf{Y} \stackrel{\text{def}}{=} \begin{bmatrix} x_{1,1}\mathbf{Y} & \cdots & x_{1,q}\mathbf{Y} \\ \vdots & \ddots & \vdots \\ x_{p,1}\mathbf{Y} & \cdots & x_{p,q}\mathbf{Y} \end{bmatrix}. \quad \square$$

Definition 1.11 (vectorization)**.** The *vec operator* vectorizes a matrix $\mathbf{X} = (x_{i,j}) \in \mathbb{R}^{p \times q}$ by stacking its columns as follows:

$$vec(\mathbf{X}) \stackrel{\text{def}}{=} [x_{1,1}, \cdots, x_{p,1}, \cdots, x_{1,q}, \cdots, x_{p,q}]^T. \quad \square$$

Tensor product ($\otimes$) and vectorization operator (vec) have the following relationship.

$$(\mathbf{B}^T \otimes \mathbf{A}) \cdot vec(\mathbf{X}) = vec(\mathbf{AXB}). \tag{1.2}$$

Definition 1.12 (matrix norm)**.** For a given $n \times n$ matrix $\mathbf{X}$, some matrix norms of $\mathbf{X}$ are defined as follows:

- max-norm: $\|\mathbf{X}\|_{\max} \stackrel{\text{def}}{=} \max_{i,j=1}^{n} |x_{i,j}|$;
- Frobenius norm: $\|\mathbf{X}\|_F \stackrel{\text{def}}{=} \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{n} |x_{i,j}|^2}$;
- 1-norm: $\|\mathbf{X}\|_1 \stackrel{\text{def}}{=} \max_{j=1}^{n} \sum_{i=1}^{n} |x_{i,j}|$;
- ∞-norm: $\|\mathbf{X}\|_\infty \stackrel{\text{def}}{=} \max_{i=1}^{n} \sum_{j=1}^{n} |x_{i,j}|$. $\square$

Definition 1.13 (spectral radius)**.** The *spectral radius* of matrix $\mathbf{X}$, denoted by $\rho(\mathbf{X})$, is the maximum of the absolute values of the eigenvalues of $\mathbf{X}$. $\square$

1.3 Challenges and Contributions

The goal of this thesis is to develop innovative techniques and novel link-based similarity models for efficiently managing relevance assessment on large and dynamic network data. In other words, we are not only interested in new techniques to tame the computational complexity for relevance assessment in a scalable fashion, but also propose novel effective models to enrich the semantics for relevance assessment, without suffering from increased computational costs. Next we enumerate the main challenges involved in the thesis.

1.3.1 All-Pairs and Bipartite SimRank

SimRank is a widely-accepted link-based similarity model, which was initially introduced by Jeh and Widom [JW02]. It is based on the philosophy that "two objects are similar if

they are referenced by similar objects". Due to its self-referentiality, fast SimRank search on large and dynamic networks poses significant challenges.

The most efficient existing approach [LVGT10] exploits partial sums memoization for computing SimRank in $O(K|V||E|)$ time on a graph $G = (V, E)$, where K is the number of iterations. However, it implies the following the limitations: (1) Although partial sums memoizing can reduce repeated calculations by caching part of similarity summations for later reuse, we observed that computations among different partial sums may have duplicate efforts. (2) For a desired accuracy ϵ, the existing SimRank model [LVGT10] requires $K = \lceil \log_C \epsilon \rceil$ iterations [LVGT10], where C is a damping factor. Nevertheless, such a geometric rate of convergence is slow in practice if a high accuracy is desirable.

In Chapter 2, we address the above issues. (1) We propose adaptive clustering strategies to eliminate partial sums redundancy, and devised novel efficient methods for speeding up the computation of SimRank to $O(Kd'|V|^2)$ time, where d' is typically much smaller than the average degree of a graph. (2) We present a new notion of SimRank that is based on a differential equation and can be represented as an exponential sum of transition matrices, as opposed to the geometric sum of the conventional counterpart. This leads to a further exponential speedup in the convergence rate of SimRank iterations. (3) In bipartite domains, we also develop a novel finer-grained partial max clustering method to speed up the computation of the Minimax SimRank variation [JW02] from $O(K|E||V|)$ to $O(K|E'||V|)$ time, where $|E'|$ $(\leq |E|)$ is the number of edges in a reduced graph after edge clustering, which can be typically much smaller than $|E|$.

Using real and synthetic data, we empirically verify that (1) our approach of partial sums sharing outperforms the best known algorithm by up to one order of magnitude. (2) The revised notion of SimRank further achieves a 5X speedup on large graphs while also fairly preserving the relative order of original SimRank scores. (3) Our finer-grained partial max memoization for the Minimax SimRank variation in bipartite domains is 0.5–1.2 orders of magnitude faster than the baselines.

1.3.2 Incremental SimRank

Real graphs are often large, and links constantly evolve with small changes over time. It is rather costly to reassess similarities of all pairs of nodes when the graph is updated. Inspired by this, we next considers fast incremental computations of SimRank on link-evolving networks.

The prior approach [LHH+10] to this issue factorizes the graph via a singular value decomposition (SVD) first, and then incrementally maintains this factorization for link updates at the expense of exactness. Consequently, all node-pair similarities are estimated in $O(r^4n^2)$ time on a graph of n nodes without guaranteed accuracy, where r is the target rank of the low-rank approximation, which is not negligibly small in practice.

In Chapter 3, we propose a novel fast incremental paradigm. (1) We characterize the SimRank update matrix $\mathbf{\Delta S}$, in response to every link update, via a rank-one Sylvester matrix equation. By virtue of this, we devise a fast incremental algorithm computing similarities of n^2 node-pairs in $O(Kn^2)$ time for K iterations. (2) We also propose an effective pruning technique capturing the "affected areas" of $\mathbf{\Delta S}$ to skip unnecessary computations, without loss of exactness. This can further accelerate the incremental SimRank computation to $O(K(nd + |\mathsf{AFF}|))$ time, where d is the average in-degree of the old graph, and $|\mathsf{AFF}|$ ($\leq n^2$) is the size of "affected areas" in $\mathbf{\Delta S}$, and in practice, $|\mathsf{AFF}| \ll n^2$.

Our empirical evaluations verify that our algorithm (a) outperforms the best known link-update algorithm [LHH+10], and (b) runs much faster than its batch counterpart when link updates are small.

1.3.3 Fast Penetrating-Rank

With the striking success of PageRank [Ber05] and SimRank [JW02], Penetrating-Rank (P-Rank) [ZHS09] has been recently proposed as another effective link-based similarity measure, since it provides a comprehensive way of encoding both incoming and outgoing links into assessment, as opposed to SimRank that considers only incoming edges for

relevance assessment. However, the existing P-Rank algorithm is iterative in nature and rather expensive to compute. Besides, accuracy estimation and stability issue for P-Rank computation have not been studied yet.

In Chapter 3, optimization techniques encompassing P-Rank accuracy, stability and computational efficiency are investigated. (1) The accuracy estimation is provided for P-Rank iterations, with the aim to find out the total number K of iterations required for achieving a desired accuracy $\epsilon > 0$. (2) A rigorous bound on the condition number of P-Rank is obtained for stability analysis. Based on this bound, it can be shown that P-Rank is stable and well-conditioned, providing that the damping factors are chosen to be suitably small. (3) Two matrix-based algorithms, applicable to digraphs and undirected graphs, are respectively devised for efficient P-Rank computation, which improves the time complexity from $O(K|V|^3)$ to $O(r^4|V|^2 + r^2|V|)$ for digraphs, and to $O(r|V|^2)$ for undirected graphs, with $|V|$ being the number of vertices in a graph, and r ($\ll |V|$) the rank of adjacency matrix. Both real and synthetic datasets are used for conducting extensive experiments to demonstrate the usefulness and efficiency of the proposed techniques for P-Rank assessment on networks.

1.3.4 Incremental Random Walk with Restart

Random Walk with Restart (RWR) is a PageRank-like object proximity model proposed by Tong and Faloutsos [TFP06]. The existing RWR model utilized a SVD method to measure object-to-object proximity in a static graph. We noticed that in practice, while edges in a graph often arrive over time, it is often cost-inhibitive to recompute proximities from scratch via *batch* algorithms when the graph is updated. This highlights the need for *incremental* algorithms to compute *changes* to the proximities in response to updates, to avoid unnecessary recomputation.

Motivated by this, in Chapter 4, we propose a fast exact incremental RWR search model over graph streams, whose edges often change over time. The most efficient method for measuring RWR proximity [FNOK12] deploys k-dash to find top-k highest proximity

nodes for a given query, which involves a strategy to incrementally *estimate* upper proximity bounds. However, due to its aim to prune needless calculation, such an incremental strategy is *approximate*: in $O(1)$ time for each node. Our main contribution for RWR is to devise an *exact* and fast incremental algorithm for edge updates. Our solution, IRWR, can incrementally compute any node proximity in $O(1)$ time for each edge update without loss of exactness. The empirical evaluations show the high efficiency and exactness of IRWR for computing proximities on dynamic networks against its batch counterparts by up to one order of magnitude. The proposed framework for assessing RWR proximities also can readily be extended to Google Personalized PageRank.

1.3.5 Fast and Incremental SimFusion

SimFusion is a very popular relevance model proposed in [XFF+05]. It has become a captivating measure of similarity between objects in a web graph. The basic concept behind SimFusion is iteratively distilled from the notion that "the similarity between two objects is reinforced by the similarity of their related objects". The existing SimFusion model [XFF+05] often leverages the *Unified Relationship Matrix* (URM) to represent latent relationships among heterogeneous data, and adopts an iterative paradigm for SimFusion computation. However, due to the row normalization of URM, we noticed that the traditional SimFusion model may produce the trivial solution, and worse still, the iterative computation of SimFusion sometimes cannot ensure the global convergence of the solution.

In Chapter 5, we propose a full treatment of SimFusion model from complexity to algorithms, aiming to support fast SimFusion search on large networks and (dynamic) graph streams. To be specific, (1) we propose SimFusion+ based on a notion of the *Unified Adjacency Matrix* (UAM), a modification of the URM, to prevent the trivial solution and the divergence issue of SimFusion. (2) We show that for any vertex-pair, SimFusion+ can be performed in $O(1)$ time and $O(|V|)$ space with an $O(K|E|)$-time precomputation done only once, as opposed to the $O(K|V|^3)$ time and $O(|V|^2)$ space of its traditional

counterpart on a graph $G = (V, E)$ for K iterations. (3) We also devise an incremental algorithm for further improving the computation of SimFusion+ when networks are dynamically updated, with performance guarantees for similarity estimation.

The experimental results on real and synthetic datasets (1) verified the scalability of the proposed SimFusion+ model, and (2) demonstrated that the proposed SimFusion+ model not only can converge to a non-trivial solution, but also allows us to identify more sensible structure information in large real-world networks.

1.3.6 Semantics Enrichment

Similarity semantics is an importance property in relevance assessment. Most recently, despite its popularity of SimRank [JW02], we observe that SimRank has an undesirable property, *i.e.,* "zero-similarity": It only accommodates paths with *equal* length from a common "center" node. Thus, a large portion of other paths are fully ignored. Similarly, RWR [TFP06] also implies a SimRank-like "zero-proximity" problem.

In Chapter 6, we attempt to remedy such issues. (1) We propose and rigorously justify SimRank*, a revised version of SimRank, which resolves such counter-intuitive "zero-similarity" problems while inheriting merits of the basic philosophy of SimRank. (2) We show that the series form of SimRank* can be reduced to a fairly succinct and elegant closed form, which looks even simpler than SimRank, yet enriches semantics without suffering from increased computational cost. This leads to a fixed-point iterative paradigm of SimRank* in $O(K|V||E|)$ time on a network $G = (V, E)$ for K iterations, which is comparable to SimRank. (3) To further optimize SimRank* computation, we leverage a novel clustering strategy via edge concentration. Due to its NP-hardness, we devise an efficient and effective heuristic to speed up SimRank* computation to $O(K|V||\tilde{E}|)$ time, where $|\tilde{E}|$ is generally much smaller than $|E|$.

The experimental evaluations, along with theoretical proofs, show that (1) SimRank* has richer semantics on real-life graphs than SimRank and RWR. This demonstrates the semantic completeness of SimRank* for similarity assessment. (2) SimRank* has higher

computational efficiency. The speedup of SimRank* on real datasets can be 5X-10X faster than SimRank and RWR.

1.4 Related Work

In this section, we briefly overview prior work in relevance assessment and related areas. Detailed comparison of previous work with specific techniques developed in this thesis will appear in the relevant chapters.

Link-based Relevance. The study of link-based relevance assessment has a long history, dating back to a series of initial papers from the early 1960's, *e.g.,* co-citation analysis [Sma73, Col74], bibliographic coupling [Kes63a, Kes63b, Kes63c], Amsler measure [Ams72], author co-citation analysis (ACA) [WG81, Eom96, McC90], co-citation proximity analysis (CPA) [GB09, Gip10], and a great deal of follow-on work, Google PageRank [Ber05, Hav03, LM03], Hyperlink-Induced Topic Search (HITS) [Kle99], SimRank [FR05, HLC+12, JW02, LLY12, LHH+10, LLY+10, LVGT10], RWR [TFP06, TFP08, TKF09], SimRank++ [AMC08], SimFusion [XFF+05], P-Rank [ZHS09] and others. Many of these previous works, especially earlier papers, focus on theoretical foundations, and not on practical considerations such as efficient relevance assessment on large networks, and incremental updates on dynamic networks, which are the important subjects of this thesis.

SimRank. SimRank is arguably one of the most successful link-based similarity measures in recent years. It was initially proposed by Jeh and Widom [JW02], who adopted an iterative paradigm to compute SimRank scores of all-pairs. Since then, there has been a surge of papers looking at various problems in efficient SimRank computing as the naive algorithm [JW02] has high time complexity. Recent results include matrix-based methods [FNSO13, LHH+10], iterative optimization [LVGT10, ZZF+13], random walk sampling [FR04, FR05, LLY12], and parallel computing [HFLC10, HLC+12]. This thesis makes a further step towards this goal by devising novel fast optimization techniques for SimRank assessment on large and dynamic networks.

RWR. RWR is another popular measure of node-to-node proximity, which is first proposed by Tong *et al.* [TFP06]. Recently, it has a board range of emerging applications, such as automatic image captioning [KSJ09], recommendation system [PYFD04, TJ13], and social networks [SBC+10, LNK03]. In addition, several other measures build upon RWR, including Personalized PageRank [Hav03], ObjectRank [HHP06], Escape Probability [TKF09], and PathSim [SHY+11]. The straightforward approach [TFP06] for computing RWR implies a matrix inversion, which is rather expensive. Recent years have witnessed growing interests in developing novel techniques to speed up RWR assessment, (*e.g.,* [FNOK12, TFP08, TKF09, SHY+11, YMS14]). While previous works mainly focus on static networks, this thesis proposes efficient incremental techniques for RWR assessment on dynamic networks.

Other Work. There has also been a large body of work for various relevance measures to serve different assessment purposes.

Personalized PageRank (PPR) [Hav03] is one of the most well-known proximity measure for ranking the importance of web pages. It is almost the same as PageRank [Ber05], except that all the random jumps are done back to the same node (not random nodes), called the "source" or "seed" node, for which we are personalizing the PageRank. Over the last decade, there are many algorithms designed to assess PPR values in different computational models, including power iteration [Hav03], approximation methods [ZFCY13, PCD+08, SBC+06], MapReduce [BCX11], and top-k search [FNS+13].

Recently, SimFusion [XFF+05] and P-Rank [ZHS09] are two appealing relevance assessment models. SimFusion is a PageRank-like relevance measure based on *similarity reinforcement assumption* that "the similarity between two data objects is reinforced by the similarity of their related objects from homogenous and heterogeneous data spaces". P-Rank [ZHS09] is a SimRank-like measure by jointly taking account of both in- and out-links for relevance assessment. The relationship between SimFusion and P-Rank was shown in [CZDC10]. Other graph relevance measure [BGH+04], role similarity [JLH11], and network clustering metric [ZCY09] are variants of SimRank and PageRank.

Chapter 2

Fast SimRank on Large Networks and Bipartite Domains

2.1 Introduction

Identifying similar objects based on link structure is a fundamental operation for many web mining tasks. Examples include web page ranking [Ber05], hypertext classification (KNN) [LLY12], graph clustering (K-means) [BC08b], and collaborative filtering [JW02]. In the last decade, with the overwhelming number of objects on the Web, there is a growing need to be able to automatically and efficiently assess their similarities on large graphs. Indeed, the Web has huge dimensions and continues to grow rapidly — more than 5% of new objects are created weekly [CR04]. As a result, similarity assessment on web objects tends to be obsolete so quickly. Thus, it is imperative to get a fast computational speed for similarity assessment on large graphs.

Amid the existing similarity metrics, SimRank [JW02] has emerged as a powerful tool for assessing structural similarities between two objects. Similar to the well-known PageRank [Ber05], SimRank scores depend merely on the Web link structure, independent of the textual content of objects. The major difference between the two models is the scoring mechanism. PageRank assigns an authority weight for each object, whereas

SimRank assigns a similarity score between two objects. SimRank was first proposed by Jeh and Widom [JW02], and has gained increasing popularity for its success in many areas such as bibliometrics [LHH+10], top-K search [LLY12], and recommender systems [AMC08]. The intuition behind SimRank is a subtle recursion that "two vertices are similar if their incoming neighbors are similar", together with the base case that "each vertex is most similar to itself" [JW02]. Due to this self-referentiality, conventional algorithms for computing SimRank have an iterative nature. The sheer size of the Web has presented striking challenges to fast SimRank computing. The best known algorithm proposed by Lizorkin *et al.* [LVGT10] (hereafter referred to as psum-SR) requires $O(Kmn)$ time ($O(Kn^3)$ in the worst case) for K iterations, where n and m denote the number of vertices and edges, respectively, in a graph.

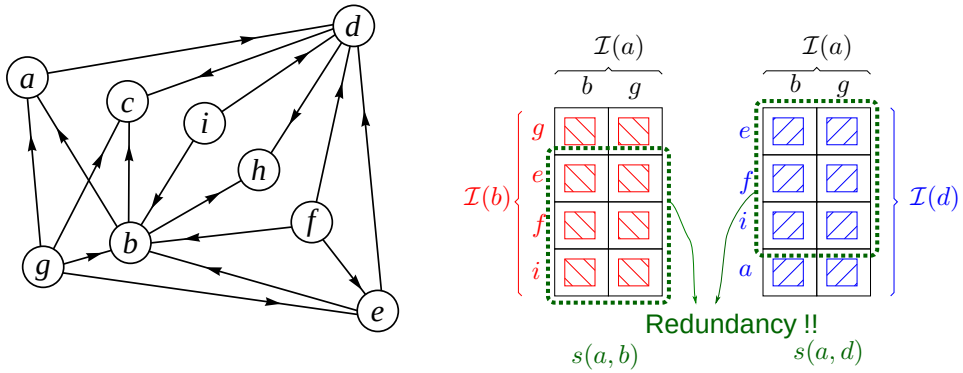
The beauty of psum-SR algorithm [LVGT10] resides in the following three observations. (1) *Essential nodes selection* may eliminate the computation of a fraction of node pairs with a-priori zero scores. (2) *Partial sums memoizing* can effectively reduce repeated calculations of the similarity among different node pairs by caching part of similarity summations for later reuse. (3) *A threshold setting* on the similarity enables a further reduction in the number of node pairs to be computed. Particularly, the second observation of *partial sums memoizing* plays a paramount role in greatly speeding up the computation of SimRank from the naive $O(Kd^2n^2)$ [JW02] to $O(Kdn^2)$, [1] where d is the number of average in-degrees in a graph.

In this chapter, we make a further step towards this goal, by proposing efficient methods for accelerating SimRank assessment on large networks and bipartite domains.

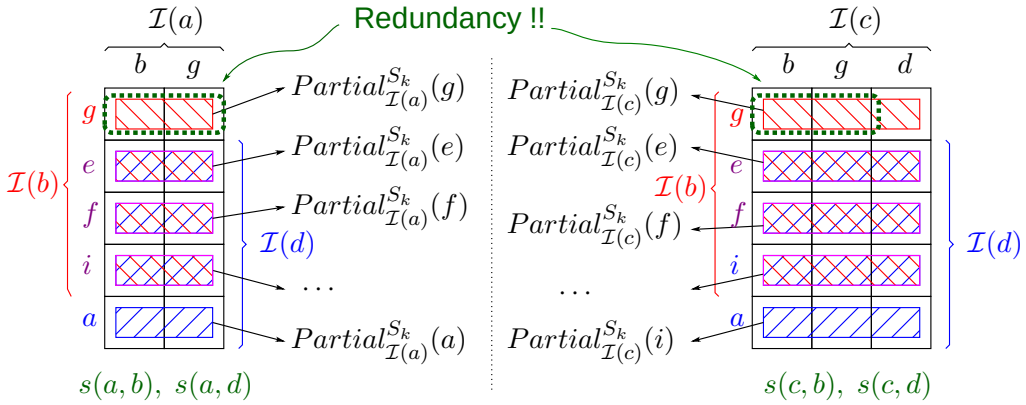
2.1.1 Motivation

Before shedding light on the blemish of psum-SR [LVGT10], let us first revisit the central idea of partial sums memoizing in Example 2.1, motivating our need to develop more efficient techniques for SimRank assessment.

[1] As $n \cdot d = m$, $O(Kmn)$ time in [LVGT10] is equivalent to $O(Kdn^2)$.



(a) A paper citation network (b) Naive method in [JW02]



(c) psum-SR leads to redundancy among different partial sums

Figure 2.1: Merits and demerits of partial sums memoization for SimRank assessment on a paper citation network

Example 2.1. Consider a paper citation network $\mathcal{G}$ in Figure 2.1a, where each vertex represents a paper, and an edge a citation. For any vertex a, we denote by $\mathcal{I}(a)$ the set of in-neighbors of a. Individual element in $\mathcal{I}(a)$ is denoted as $\mathcal{I}_i(a)$. Let $s(a,b)$ be the SimRank similarity between vertices a and b. In what follows, we want to compute $s(a,b)$ and $s(a,d)$ in $\mathcal{G}$.

Before partial sums memoizing is introduced, a naive way is to sum up the similarities of all in-neighbors $(\mathcal{I}_i(a), \mathcal{I}_j(b))$ of (a,b) for computing $s(a,b)$, and to sum up the similarities of all in-neighbors $(\mathcal{I}_i(a), \mathcal{I}_j(d))$ of (a,d) for computing $s(a,d)$, *independently*, as depicted in Figure 2.1b. In contrast, psum-SR is based on the observation that $\mathcal{I}(b)$

and $\mathcal{I}(d)$ have three vertices $\{e, f, i\}$ in common. Thus, the three partial sums over $\mathcal{I}(a)$ (*i.e.,* $Partial^{s_k}_{\mathcal{I}(a)}(y)$ [2] with $y \in \{e, f, i\}$) can be computed only once, and reused for both $s(a, b)$ and $s(a, d)$ computation (see left part of Figure 2.1c). Similarly, for computing $s(c, b)$ and $s(c, d)$, since $\mathcal{I}(b) \cap \mathcal{I}(d) = \{e, f, i\}$, the partial sums over $\mathcal{I}(c)$ (*i.e.,* $Partial^{s_k}_{\mathcal{I}(c)}(x)$ with $x \in \{e, f, i\}$) can be cached for later reuse (see right part of Figure 2.1c). □

Despite the aforementioned merits of psum-SR, the existing work [LVGT10] on SimRank has the following limitations.

Firstly, we observe from Example 2.1 that computing partial sums [LVGT10] over different in-neighbor sets may have duplicate redundancy. For instance, $\mathcal{I}(a)$ and $\mathcal{I}(c)$ in Figure 2.1c have two vertices $\{b, g\}$ in common, implying that the sub-summation $Partial^{s_k}_{\{b,g\}}(\star)$ is the common part shared between the partial sums $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ and $Partial^{s_k}_{\mathcal{I}(c)}(\star)$. Thus, there is an opportunity to speed up the computation of SimRank by preprocessing the common sub-summation $Partial^{s_k}_{\{b,g\}}(\star)$ once, and caching it for both $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ and $Partial^{s_k}_{\mathcal{I}(c)}(\star)$ computation. However, it is a big challenge to identify the well-tailored common parts for maximal sharing among the partial sums over different in-neighbor sets since there could be many irregularly and arbitrarily overlapped in-neighbor sets in a real graph. To address this issue, we propose optimization techniques to have such common parts memoized in a hierarchical clustering manner, and devise an efficient algorithm to eliminate such redundancy.

Secondly, the existing iterative paradigm [LVGT10] for computing SimRank has a geometric rate of convergence, which might be, in practice, rather slow when a high accuracy is attained. This is especially evident in *e.g.,* citation networks and web graphs [KNT06]. For instance, our experiments on DBLP citation network shows that a desired accuracy of $\epsilon = 0.001$ may lead to more than 30 iterations of SimRank, for the damping

[2] Recall from [LVGT10] that *a partial sum* for a binary function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ over a set $\mathcal{D} = \{x_1, \cdots, x_n\} \subseteq \mathcal{X}$, denoted by $Partial^f_{\mathcal{D}}(\star)$, is defined as

$$Partial^f_{\mathcal{D}}(y) = \sum_{x_i \in \mathcal{D}} f(x_i, y), \quad (y \in \mathcal{Y}).$$

factor $C = 0.8$. Lizorkin *et al.* has proved theoretically in [LVGT10] that, for a desired accuracy ϵ, the number of iterations required for the conventional SimRank is $K = \lceil \log_C \epsilon \rceil$, which is mainly due to the geometric sum of the traditional representation of SimRank. This highlights the need for a revised SimRank model to speed up the geometric rate of convergence.

Moreover, for bipartite domains, a variant model of SimRank proposed by Jeh and Widom in [JW02, Section 4.3.2], called the Minimax Variation SimRank, may also have duplicate computational efforts in computing the partial *max* over every out-neighbor set for all vertex-pair similarities. However, we observe that the choices of granularity for partial *max* memoization is different from those for partial *sums* memoization. This is because, in the context of partial *sums* sharing, "subtraction" is allowed to compute one partial sum from another, whereas, in the context of partial *max* sharing, "subtraction" is disallowed. We will provide a detailed discussion in Section 2.5.

2.1.2 Chapter Outlines

In this chapter, our main contributions are summarized as follow.

- We propose an adaptive clustering strategy based on a minimum spanning tree to eliminate duplicate computations in partial sums [LVGT10] in a hierarchical fashion (Section 2.3). By optimizing the sub-summations sharing among different partial sums, an efficient algorithm is devised for speeding up the computation of SimRank from $O(Kdn^2)$ [LVGT10] to $O(Kd'n^2)$ time, where d' ($\leq d$) can, in general, be much smaller than the average in-degree d.

- We introduce a new notion of SimRank by using a matrix differential equation to further accelerate the convergence of SimRank iterations from the original geometric to exponential rate (Section 2.4). We show that the new notion of SimRank can be characterized as an exponential sum in terms of the transition matrix while fairly preserving the relative order of SimRank, as opposed to the conventional

counterpart [LVGT10] as a geometric sum. We also devise a space-efficient iterative paradigm for computing the differential SimRank matrix equation, which can integrate our previous techniques of sub-summations sharing without sacrificing extra memory space.

- We investigate the partial max sharing problem for speeding up the computation of the Minimax SimRank variation in bipartite graphs, a variant model proposed in [JW02, Section 4.3.2]. We show that the partial *max* sharing problem is different from the partial *sums* sharing problem, due to "subtraction" curse in the context of max operator. To resolve this issue, we devise a novel finer-grained partial *max* clustering strategy via edge concentration, improving the computation of Minimax SimRank variation from $O(Kmn)$ to $O(Km'n)$ time, where m' ($\leq m$) is the number of edges in a reduced graph after edge clustering, which is practically smaller than m (Section 2.5).

- We conduct extensive experiments on real and synthetic datasets (Section 2.6), demonstrating that (1) our approach of partial sum sharing on large graphs can be one order of magnitude faster than psum-SR; (2) our revised notion of SimRank achieves up to a 5X further speedup against the conventional counterpart; and (3) for the Minimax SimRank variation in bipartite domains, our finer-grained partial max sharing method outperforms the baselines by 0.5–1.2 orders of magnitude in computational time.

2.2 Preliminaries

We revisit the two forms of SimRank, To our knowledge, there are two representations of SimRank, *i.e.,* the iterative form [JW02, LVGT10], and matrix form [LHH+10, HFLC10]. The consistency of two forms was pointed out in [LHH+10].

2.2.1 Iterative Form

Recall from Definition 1.7 of SimRank in Chapter 1. The SimRank formula (1.1) naturally leads to the following iterative method [JW02]:

Start with $s_0(a,b) = \begin{cases} 1, & a=b; \\ 0, & a \neq b. \end{cases}$, and for $k = 0, 1, \cdots$, set

(i) $s_{k+1}(a,a) = 1$;

(ii) $s_{k+1}(a,b) = 0$, if $\mathcal{I}(a) = \varnothing$ or $\mathcal{I}(b) = \varnothing$;

(iii) otherwise,

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} s_k(i,j). \tag{2.1}$$

The resultant sequence $\{s_k(a,b)\}_{k=0}^{\infty}$ converges to $s(a,b)$, the *exact* solution of Eq.(1.1).

2.2.2 Matrix Form

In matrix notations [LHH+10], SimRank can be formulated as

$$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n, \tag{2.2}$$

where $\mathbf{S}$ is the similarity matrix whose entry $[\mathbf{S}]_{a,b}$ is the similarity score $s(a,b)$, $\mathbf{Q}$ is the backward transition matrix whose entry $[\mathbf{Q}]_{a,b} = \frac{1}{|\mathcal{I}(a)|}$ if there is an edge from b to a, and 0 otherwise, and $\mathbf{I}_n$ is an $n \times n$ identity matrix.

2.3 Eliminating Partial Sums Duplicate Computations

The existing method, psum-SR [LVGT10], of performing Eq.(2.1) is to memoize the partial sums over $\mathcal{I}(a)$ first:

$$Partial^{s_k}_{\mathcal{I}(a)}(j) = \sum_{i \in \mathcal{I}(a)} s_k(i,j), \quad (\forall j \in \mathcal{I}(b)) \tag{2.3}$$

and then iteratively compute $s_{k+1}(a,b)$ as follows:

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} Partial^{s_k}_{\mathcal{I}(a)}(j). \tag{2.4}$$

Consequently, the results of $Partial^{s_k}_{\mathcal{I}(a)}(j)$, $\forall j \in \mathcal{I}(b)$, can be reused later when we compute the similarities $s_{k+1}(a, \star)$ for a given vertex a as the first argument. However, we observe that the partial sums over different in-neighbor sets may share common sub-summations. For example in Figure 2.1c, the partial sums $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ and $Partial^{s_k}_{\mathcal{I}(c)}(\star)$ have the sub-summation $Partial^{s_k}_{\{b,g\}}(\star)$ in common. By virtue of this, we next show how to optimize sub-summations sharing among different partial sums.

2.3.1 Partition In-neighbor Sets for Inner Partial Sums Sharing

We first introduce the notion of *a set partition*.

Definition 2.2. *A partition* of a set $\mathcal{D}$, denoted by $\mathscr{P}(\mathcal{D})$, is a family of disjoint subsets $\mathcal{D}_i$ of $\mathcal{D}$ whose union is $\mathcal{D}$:

$$\mathscr{P}(\mathcal{D}) = \{\mathcal{D}_1, \mathcal{D}_2, \cdots, \mathcal{D}_p\}, \text{ with } p = |\mathscr{P}(\mathcal{D})|,$$

where $\mathcal{D}_i \cap \mathcal{D}_j = \varnothing$ for $i \neq j$, and $\bigcup_{i=1}^{p} \mathcal{D}_i = \mathcal{D}$. □

For instance, $\mathscr{P}(\mathcal{I}(b)) = \{\{f, g\}, \{e, i\}\}$ is a partition of the in-neighbor set $\mathcal{I}(b) = \{f, g, e, i\}$ in Figure 2.1a.

The set partition is deployed for speeding up SimRank computation, based on the proposition below.

Proposition 2.3. For two distinct vertices a and b with $\mathcal{I}(a) \neq \varnothing$ and $\mathcal{I}(b) \neq \varnothing$, $s_{k+1}(a, b)$ can be iteratively computed as

$$s_{k+1}(a, b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{\Delta \in \mathscr{P}(\mathcal{I}(a))} Partial^{s_k}_{\Delta}(j). \tag{2.5}$$

Here, $Partial^{s_k}_{\Delta}(j)$ is defined as Eq.(2.3) with $\mathcal{I}(a)$ replaced by Δ. □

Sketch of Proof. The proof follows immediately from the following two facts:

(i) For two disjoint sets $\mathcal{A}$ and $\mathcal{B}$, $Partial^{s_k}_{\mathcal{A}}(j) + Partial^{s_k}_{\mathcal{B}}(j) = Partial^{s_k}_{\mathcal{A} \cup \mathcal{B}}(j)$, $\forall j$.

(ii) $\bigcup_{\Delta \in \mathscr{P}(\mathcal{I}(a))} = \mathcal{I}(a)$, $\forall a \in \mathcal{V}$. □

The main idea in our approach is to share the common sub-summations among different partial sums, by precomputing the sub-summations $Partial^{s_k}_{\Delta}(\star)$ over $\Delta \in \mathscr{P}(\mathcal{I}(a))$ once, and caching them in a block fashion for later reuse, which can effectively avoid repeating duplicate sub-summations. As an example in Figure 2.1c, when $\mathcal{I}(c)$ is partitioned as $\mathscr{P}(\mathcal{I}(c)) = \{\mathcal{I}(a), \{d\}\}$ with $\mathcal{I}(a) = \{b, g\}$, once computed, the sub-summations $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ can be memoized and reused for computing $Partial^{s_k}_{\mathcal{I}(c)}(\star)$. In contrast, the existing method psum-SR [LVGT10] has to start from scratch to compute $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ and $Partial^{s_k}_{\mathcal{I}(c)}(\star)$, independently, which is due to no reuse of common sub-summations.

The selection of a partition $\mathscr{P}(\mathcal{I}(a))$ for an in-neighbor set $\mathcal{I}(a)$ has a great impact on the performance of our approach. Troubles could be expected when a selected partition $\mathscr{P}(\mathcal{I}(a))$ is too coarse or too fine. For instance, if $\mathcal{I}(a)$ is taken to be a trivial partition of itself, *i.e.,* $\mathscr{P}(\mathcal{I}(a)) = \{\mathcal{I}(a)\}$ for every vertex a, Eq.(2.5) can be simplified to the conventional psum-SR iteration in Eq.(2.4). From this perspective, our approach is a generalization of psum-SR. On the other hand, if the partitions of $\mathcal{I}(a)$ become finer (*i.e.,* the size of $\Delta \in \mathscr{P}(\mathcal{I}(a))$ becomes smaller), there is a more likelihood of $Partial^{s_k}_{\Delta}(\star)$ with a high density of common sub-summations, but with a low cardinality on the similarity values to be clustered. An extreme example would be a discrete partition of $\mathcal{I}(a)$, *i.e.,* $\mathscr{P}(\mathcal{I}(a)) = \{\{x\} | x \in \mathcal{I}(a)\}$, where every block is a singleton vertex. In such a case, Eq.(2.5) would deteriorate to the naive iteration [JW02] in Eq.(2.1), which may be even worse than psum-SR. Thus, it is desirable to find the best partition $\mathscr{P}(\mathcal{I}(a))$ for each $\mathcal{I}(a)$ that has the largest and densest clumps of common vertices.

The problem of finding such optimal partitions to minimize the total cost of partial sums over different in-neighbor sets, referred to as *Optimal In-neighbors Partitioning* and denoted as OIP, can be formulated as follows:

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, OIP is to find the optimal partition $\mathscr{P}(\mathcal{I}(a)) = \{\Delta^i_a \mid i = 1, \cdots, |\mathscr{P}(\mathcal{I}(a))|\}$ of each in-neighbor set $\mathcal{I}(a)$, $a \in \mathcal{V}$, for creating chunks Δ^i_a such that the total number of additions required for computing all the partial sums $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ over every in-neighbor set $\mathcal{I}(a)$, $a \in \mathcal{V}$, is minimized by reusing the sub-summation results

$Partial^{s_k}_{\Delta^i_a}(\star)$ over chunks Δ^i_a.

Proposition 2.4. The OIP problem is NP-hard. □

Proof. We verify this by reducing the NP-complete *Ensemble Computation* (EC) problem [GJ79, p.66] to a special case of the decision problem of OIP.

The EC problem is defined as follows: Given a collection $\mathscr{C}$ of subsets of a finite set $\mathcal{A}$ and a positive integer J, EC is to decide whether there is a sequence $(z_1 = x_1 \cup y_1, \cdots, z_j = x_j \cup y_j)$ of $j \leq J$ union operations, where each x_i and y_i is either $\{a\}$ for some $a \in \mathcal{A}$ or z_p for some $p < i$, such that x_i and y_i are disjoint for $1 \leq i \leq j$ and such that for every subset $\mathcal{C} \in \mathscr{C}$ there is some z_i, $1 \leq i \leq j$, that is identical to $\mathcal{C}$.

For each instance of EC, we construct the corresponding instance of the OIP decision problem by setting $\mathcal{A} = \{s_k(a, \star) \mid a \in \mathcal{V}\}$, $\mathscr{C} = \{Partial^{s_k}_{\mathcal{I}(a)}(\star) \mid a \in \mathcal{V}\}$, and an integer J to be the maximum number of required additions. Clearly, by converting union operations ($\cup$) of EC into additions ($+$), it follows that the OIP decision problem has a solution, *i.e.,* $\exists$ a sequence $(z_1 = x_1 + y_1, \cdots, z_j = x_j + y_j)$ of $j \leq J$ additions, if and only if there exists a sequence $(z_1 = x_1 \cup y_1, \cdots, z_j = x_j \cup y_j)$ of $j \leq J$ union operations for EC. Thus, the NP-completeness of the OIP decision problem follows immediately from the NP-completeness of EC.

Also, the decision problem of OIP can be naturally converted into its corresponding optimization problem by imposing a bound on the number of additions to be optimized, namely, turning "whether there exists such a solution that can be done in fewer than J additions" into "minimize the number of additions". Hence, the OIP optimization problem is NP-hard due to the NP-completeness of its decision problem. □

We next seek for a good heuristic method for OIP.

The basic idea is as follows. Consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. For every two in-neighbor sets $\mathcal{I}(a)$ and $\mathcal{I}(b)$ of vertices $a, b \in \mathcal{V}$, we first calculate *the transition cost*

from $\mathcal{I}(a)$ to $\mathcal{I}(b)$, denoted by $\mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)}$, as follows: [3]

$$\mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)} \triangleq \min\{|\mathcal{I}(a) \ominus \mathcal{I}(b)|, |\mathcal{I}(b)| - 1\}, \tag{2.6}$$

where $\ominus$ is the *symmetric difference* of two sets. [4] Thus, the value of $\mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)}$ is actually the number of additions required to compute the partial sum $Partial^{s_k}_{\mathcal{I}(b)}(\star)$, given the partial sum $Partial^{s_k}_{\mathcal{I}(a)}(\star)$. Then, we construct a weighted digraph $\mathscr{G} = (\mathscr{V}, \mathscr{E})$ whose vertices correspond to the non-empty in-neighbor sets of $\mathcal{G}$, with an extra vertex corresponding to an empty set $\varnothing$, *i.e.*, $\mathscr{V} = \{\mathcal{I}(a) \mid a \in \mathcal{V}\} \cup \{\varnothing\}$. There is an edge from $\mathcal{I}(a)$ to $\mathcal{I}(b)$ in $\mathscr{G}$ if $|\mathcal{I}(a)| \leq |\mathcal{I}(b)|$. The weight of an edge $(\mathcal{I}(a), \mathcal{I}(b)) \in \mathscr{E}$ represents the transition cost $\mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)}$. Finally, we find a minimum spanning tree of $\mathscr{G}$, denoted by $\mathscr{T}$, whose total transition cost is minimum. Henceforth, every edge $(\mathcal{I}(a), \mathcal{I}(b))$ in $\mathscr{T}$ implies the following: (i) $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ should be computed prior to $Partial^{s_k}_{\mathcal{I}(b)}(\star)$ computation, which provides an optimized topological sort for efficiently computing all the partial sums. (ii) $\mathcal{I}(b)$ needs to be partitioned as $\mathcal{I}(b) \cap \mathcal{I}(a)$ and $\mathcal{I}(b) \backslash \mathcal{I}(a)$, meaning that the result of $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ can be cached and shared with $Partial^{s_k}_{\mathcal{I}(b)}(\star)$ computation.

The following example depicts how this idea works.

Example 2.5. Consider the network $\mathcal{G}$ in Figure 2.1a, with the vertices and the corresponding non-empty in-neighbor sets depicted in Figure 2.2a. We show how to find a decent ordering for partial sums computing and sharing in $\mathcal{G}$.

Firstly, we compute the transition cost of each pair of in-neighbor sets (along with an empty set $\varnothing$) in $\mathcal{G}$, by using Eq.(2.6). The results are shown in Figure 2.2b, where each cell describes the transition cost from the in-neighbor set in the left most column

[3] Without loss of generality, only in the case of $|\mathcal{I}(a)| \leq |\mathcal{I}(b)|$, we need to compute $\mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)}$. This is because we are interested only in the cost of computing $Partial^{s_k}_{\mathcal{I}(b)}(\star)$ by using the given $Partial^{s_k}_{\mathcal{I}(a)}(\star)$. Conversely, if utilizing the result of $Partial^{s_k}_{\mathcal{I}(b)}(\star)$ to compute $Partial^{s_k}_{\mathcal{I}(a)}(\star)$, for $|\mathcal{I}(a)| \leq |\mathcal{I}(b)|$, then we have to introduce the "subtraction" to undo the summation that we have already done, which is often an extra operation.

[4] The *symmetric difference* of two sets $\mathcal{A}$ and $\mathcal{B}$, denoted by $\mathcal{A} \ominus \mathcal{B}$, is the set of all elements of $\mathcal{A}$ or $\mathcal{B}$ which are not in both $\mathcal{A}$ and $\mathcal{B}$. Symbolically,

$$\mathcal{A} \ominus \mathcal{B} = (\mathcal{A} \backslash \mathcal{B}) \cup (\mathcal{B} \backslash \mathcal{A}).$$

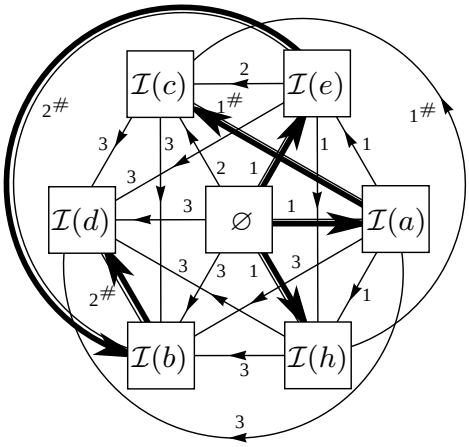
As an example in Fig 2.1c, given $\mathcal{I}(b) = \{g, e, f, i\}$ and $\mathcal{I}(d) = \{e, f, i, a\}$, we have $\mathcal{I}(b) \ominus \mathcal{I}(d) = \{g, a\}$.

vertex	$\mathcal{I}(\star)$
a	$\{b, g\}$
e	$\{f, g\}$
h	$\{b, d\}$
c	$\{b, d, g\}$
b	$\{f, g, e, i\}$
d	$\{f, a, e, i\}$

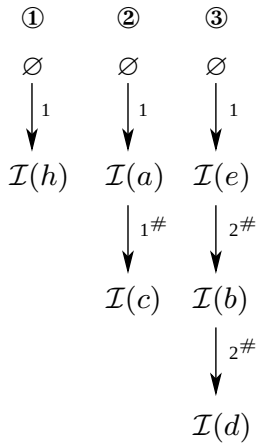
(a) In-neighbors in $\mathcal{G}$

	$\mathcal{I}(a)$	$\mathcal{I}(e)$	$\mathcal{I}(h)$	$\mathcal{I}(c)$	$\mathcal{I}(b)$	$\mathcal{I}(d)$
$\varnothing$	1	1	1	2	3	3
$\mathcal{I}(a)$		1	1	$1^{\#}$	3	3
$\mathcal{I}(e)$			1	2	$2^{\#}$	3
$\mathcal{I}(h)$				$1^{\#}$	3	3
$\mathcal{I}(c)$					3	3
$\mathcal{I}(b)$						$2^{\#}$

(b) Transition Costs (Edge Weights) in $\mathscr{G}$



(c) Minimum Spanning Tree $\mathscr{T}$



(d) Topological Sort

Figure 2.2: Constructing a minimum spanning tree $\mathscr{T}$ to find an optimized topological sort for partial sums sharing

to the in-neighbor set in the top line. For instance, the cell '$2^{\#}$' at row '$\mathcal{I}(e)$' column '$\mathcal{I}(b)$' shows that $\mathcal{TC}_{\mathcal{I}(e)\to\mathcal{I}(b)} = 2$. This cell is tagged with #, indicating that the partial sum $Partial^{s_k}_{\mathcal{I}(b)}(\star)$ can be computed from the memoized result of $Partial^{s_k}_{\mathcal{I}(e)}(\star)$ (rather than from scratch). This is because the transition cost 2 is, in essence, obtained from the 2 operations of symmetric difference (*i.e.*, $|\mathcal{I}(e) \ominus \mathcal{I}(b)| = |\{e, i\}| = 2$) in lieu of the 3 additions (*i.e.*, $|\mathcal{I}(b)| - 1 = 3$) *w.r.t.* Eq.(2.6). Note that the lower triangular part of the table in Figure 2.2b remains empty since we are interested only in the cost $\mathcal{TC}_{\mathcal{I}(x)\to\mathcal{I}(y)}$ when $|\mathcal{I}(x)| \le |\mathcal{I}(y)|$.

Next, we build a weighted digraph $\mathscr{G}$ in Figure 2.2c, with vertices corresponding to the non-empty in-neighbor sets (plus $\varnothing$) of $\mathcal{G}$ (which are in column '$\mathcal{I}(\star)$' of Figure 2.2a), and edge weights to the transition costs. For instance, the weight of the edge $(\mathcal{I}(e), \mathcal{I}(b))$

in $\mathscr{G}$ is associated with the cell '$2^{\#}$' at row '$\mathcal{I}(e)$' column '$\mathcal{I}(b)$' in Figure 2.2b. Thus, every path in $\mathscr{G}$ yields a linear ordering of partial sums computation. More importantly, partial sums sharing may occur in the edges tagged with #. As an example, the path $\varnothing \xrightarrow{1} \mathcal{I}(e) \xrightarrow{2^{\#}} \mathcal{I}(b)$ in $\mathscr{G}$ shows that (i) $Partial^{s_k}_{\mathcal{I}(e)}(\star)$ is computed from scratch (from $\varnothing$) with 1 operation, and (ii) $Partial^{s_k}_{\mathcal{I}(b)}(\star)$ is obtained by reusing the result of $Partial^{s_k}_{\mathcal{I}(e)}(\star)$, involving 2 operations.

Finally, we find a directed minimum spanning tree $\mathscr{T}$ of $\mathscr{G}$, by starting from the vertex $\varnothing$, and choosing the cheapest path for partial sums computing and sharing, as depicted in bold edges in Figure 2.2c. Consequently, using depth-first search (DFS), we can obtain 3 paths from $\mathscr{T}$ for partial sums optimization, as shown in Figure 2.2d.

Using this idea, we can identify the moderate partitions of each in-neighbor set in $\mathcal{G}$, with large and dense chunks for sub-summations sharing. Such partitions are not optimal, but can, in practice, achieve better performances than psum-SR. Proposition 2.6 shows the correctness.

Proposition 2.6. Given two distinct non-empty in-neighbor sets $\mathcal{I}(a)$ and $\mathcal{I}(b)$, and a partial sum $Partial^{s_k}_{\mathcal{I}(a)}(\star)$, if $|\mathcal{I}(a) \ominus \mathcal{I}(b)| < |\mathcal{I}(b)| - 1$, then we have the following:

(i) $\mathcal{I}(b)$ can be partitioned as

$$\mathcal{I}(b) = (\mathcal{I}(b) \cap \mathcal{I}(a)) \cup (\mathcal{I}(b) \backslash \mathcal{I}(a)). \tag{2.7}$$

(ii) The partial sum $Partial^{s_k}_{\mathcal{I}(b)}(\star)$ can be computed from the cached result of $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ as follows:

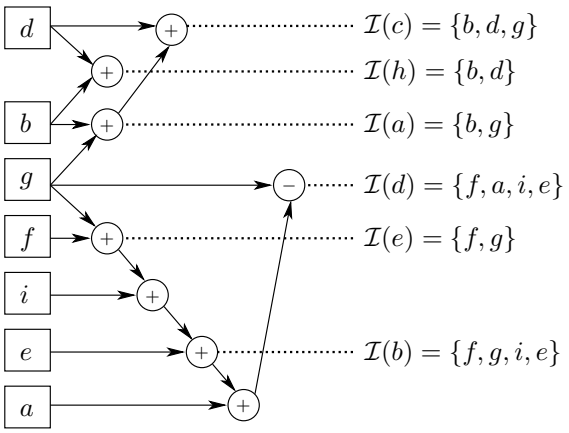
$$\begin{aligned} Partial^{s_k}_{\mathcal{I}(b)}(y) = Partial^{s_k}_{\mathcal{I}(a)}(y) &- \sum_{x \in \mathcal{I}(a) \backslash \mathcal{I}(b)} s_k(x, y) \\ &+ \sum_{x \in \mathcal{I}(b) \backslash \mathcal{I}(a)} s_k(x, y), \quad (y \in \mathcal{V}) \end{aligned} \tag{2.8}$$

with $|\mathcal{I}(a) \ominus \mathcal{I}(b)|$ operations being performed. □

Sketch of Proof. The proof of Eq.(2.7) is trivial, whereas the proof of Eq.(2.8) is based on two facts :

	$\mathscr{P}(\star)$
$\mathcal{I}(a)$	$\{\{b,g\}\}$
$\mathcal{I}(e)$	$\{\{f,g\}\}$
$\mathcal{I}(h)$	$\{\{b,d\}\}$
$\mathcal{I}(c)$	$\{\mathcal{I}(a),\{d\}\}$
$\mathcal{I}(b)$	$\{\mathcal{I}(e),\{e,i\}\}$
$\mathcal{I}(d)$	$\{\mathcal{I}(b)\backslash\{g\},\{a\}\}$

(a) Partitions of $\mathcal{I}(\star)$ in $\mathcal{G}$



(b) Hierarchical Clustering

Figure 2.3: In-neighbor sets partitioning dendrogram

(i) $\mathcal{B} = (\mathcal{A}\backslash(\mathcal{A}\backslash\mathcal{B})) \cup (\mathcal{B}\backslash\mathcal{A})$;

(ii) $Partial^{s_k}_{\mathcal{A}\backslash\mathcal{B}}(j) = Partial^{s_k}_{\mathcal{A}}(j) - Partial^{s_k}_{\mathcal{B}\cap\mathcal{A}}(j),\ \forall j.$ □

In the sequel, we give an illustrative example to show how to find all the partitions of in-neighbor sets for partial sums sharing via Proposition 2.6.

Example 2.7 (Find all the partitions of in-neighbor sets for partial sums sharing)**.** Recall the network $\mathcal{G}$ in Figure 2.1a, along with the optimized ordering of partial sums in Figure 2.2d. We show how to identify the partition of each in-neighbor set in $\mathcal{G}$ for partial sums sharing. For instance, consider the path $\varnothing \xrightarrow{1} \mathcal{I}(a) \xrightarrow{1^{\#}} \mathcal{I}(c)$ in Figure 2.2d. We have the following.

(i) The first edge $\varnothing \xrightarrow{1} \mathcal{I}(a)$ implies that $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ need to be computed from scratch since the starting point of this edge is $\varnothing$. Thus, $\mathcal{I}(a)$ has only one partition of itself.

(ii) The second edge $\mathcal{I}(a) \xrightarrow{1^{\#}} \mathcal{I}(c)$ suggests that $\mathcal{I}(c)$ can be partitioned, by using Eq.(2.7), as

$$\mathcal{I}(c) = (\mathcal{I}(c) \cap \mathcal{I}(a)) \cup (\mathcal{I}(c)\backslash\mathcal{I}(a)) = \mathcal{I}(a) \cup \{d\}.$$

Hence, $Partial^{s_k}_{\mathcal{I}(c)}(\star)$ can be obtained from the memoized result of $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ via

Eq.(2.8) as follows:

$$Partial^{s_k}_{\mathcal{I}(c)}(x) = Partial^{s_k}_{\mathcal{I}(a)}(x) + s_k(d, x). \quad (x \in \mathcal{V})$$

We repeat these steps for the rest of two paths in Figure 2.2d. Finally, we get all the partitions of in-neighbor sets in $\mathcal{G}$, as shown in Figure 2.3a. Accordingly, the resultant accumulation of reusable partial sums is visualized in Figure 2.3b, in which a letter with a box denotes a vertex, and a symbol with a circle an operator. For example, '$\boxed{d} \oplus \boxed{b} \cdots \mathcal{I}(h)$' means that $s_k(d, \star)$ and $s_k(b, \star)$ are added to yield $Partial^{s_k}_{\mathcal{I}(h)}(\star)$. □

2.3.2 Use In-neighbor Set Partitions for Outer Sums Sharing

After the partitions of in-neighbor sets have been identified for *(inner)* partial sums sharing, optimization approaches in this subsection allow *outer* partial sums sharing for further speeding up the computation of SimRank.

To avoid ambiguity, we refer to the sums *w.r.t.* the index i in Eq.(2.3) as *(inner) partial sums*, and the sums *w.r.t.* the index j in Eq.(2.4) as *outer partial sums*.

Our key observation is as follows. Recall from Eq.(2.4) that, given the memoized results of partial sums $Partial^{s_k}_{\mathcal{I}(a)}(\star)$, the existing algorithm psum-SR for computing $s_k(a, b)$ is to sum up $Partial^{s_k}_{\mathcal{I}(a)}(y)$, one by one, over all $y \in \mathcal{I}(b)$. Such a process can be pictorially depicted in the left part of Figure 2.1c, in which each horizontal bar represents a partial sum over $\mathcal{I}(a)$. In order to compute $s(a, b)$, we need to add up the horizontal bars (*i.e.,* the partial sums) in the first four rows. However, while computing $s(a, c)$ by adding up the horizontal bars in the last four rows, we observe that the three horizontal bars at rows 'e','f','i' may suffer from repetitive additions. As another example in the right part of Figure 2.1c, for computing $s(b, c)$ and $s(d, c)$, the sum of the three horizontal bars at rows 'e','f','i' is again a repeated operation. As such, the major problem of Eq.(2.4) is the one-by-one fashion in which the partial sums $Partial^{s_k}_{\mathcal{I}(a)}(y)$ for $y \in \mathcal{I}(b)$ are added together.

Our main idea in optimizing Eq.(2.4) is to split $\mathcal{I}(b)$ into several chunks Δ^i_b first, such

that

$$\mathscr{P}(\mathcal{I}(b)) = \{\Delta_b^i \mid i = 1, \cdots, |\mathscr{P}(\mathcal{I}(b))|\},$$

and then add up the cached results of partial sums in a chunk-by-chunk fashion to compute $s_{k+1}(a,b)$ as

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{\Delta_b^i \in \mathscr{P}(\mathcal{I}(b))} OuterPartial_{\Delta_b^i}^{\mathcal{I}(a),s_k} \tag{2.9}$$

with

$$OuterPartial_{\Delta_b^i}^{\mathcal{I}(a),s_k} \triangleq \sum_{j \in \Delta_b^i} Partial_{\mathcal{I}(a)}^{s_k}(j).$$

In contrast with Eq.(2.4), our method in Eq.(2.9) can eliminate the redundancy among different outer partial sums. Once computed, the outer partial sum $OuterPartial_{\Delta_b^i}^{\mathcal{I}(a),s_k}$ is memoized and can be reused later without recalculation again. As an example in Figure 2.1c, suppose $\mathcal{I}(b)$ and $\mathcal{I}(d)$ are split into

$$\mathcal{I}(b) = \{g\} \cup \{e,f,i\}, \quad \mathcal{I}(d) = \{e,f,i\} \cup \{a\},$$

the outer partial sum $OuterPartial_{\{e,f,i\}}^{\mathcal{I}(a),s_k}$ is computed only once and can be reused in both $s_{k+1}(a,b)$ and $s_{k+1}(a,d)$ computation.

The problem of finding an ideal partition $\mathscr{P}(\mathcal{I}(b))$ of $\mathcal{I}(b)$ for maximal sharing outer partial sums is still NP-hard, and its proof is the same as that of OIP in Proposition 2.4. Thus, the partitioning techniques for (inner) partial sums sharing in Subsection 2.3.1 can be applied in a similar way to optimize outer partial sums sharing. In other words, the partitions of in-neighbor sets in Eq.(2.7) for (inner) partial sums sharing, once identified, can be reused later for outer partial sums sharing. The correctness is verified in Proposition 2.8.

Proposition 2.8. Given two non-empty in-neighbor sets $\mathcal{I}(b)$ and $\mathcal{I}(d)$, an outer partial sum $OuterPartial_{\mathcal{I}(b)}^{\mathcal{I}(a),s_k}$, and (inner) partial sums $Partial_{\mathcal{I}(a)}^{s_k}(\star)$, if $|\mathcal{I}(b) \ominus \mathcal{I}(d)| < |\mathcal{I}(d)| - 1$, then we have the following:

vertex	$Partial^{s_k}_{\mathcal{I}(x)}(y)$			$OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(z)}$		$s_{k+1}(x,z)$	
x	$y=b$	$y=g$	$y=d$	$z=a$	$z=c$	$z=a$	$z=c$
a	1	1	0.11	2	2.11	1	0.21
e	0	1	0	1	1	0.15	0.1
h	1.11	0	1.11	1.11	2.22	0.17	0.22
c	1.11	1	1.11	2.11	3.22	0.21	1
b	0.15	1	0.08	1.15	1.23	0.09	0.06
d	0.23	0	0.08	0.23	0.31	0.02	0.02

Figure 2.4: Computing $s_{k+1}(x,a)$ and $s_{k+1}(x,c)$, $\forall x \in \mathcal{V}$, by using outer sums sharing ($k=2$ and $C=0.6$)

(i) $OuterPartial^{\mathcal{I}(a),s_k}_{\mathcal{I}(d)}$ can be computed from the memoized results of $OuterPartial^{\mathcal{I}(a),s_k}_{\mathcal{I}(b)}$, $\forall a \in \mathcal{V}$, as follows:

$$OuterPartial^{\mathcal{I}(a),s_k}_{\mathcal{I}(d)} = OuterPartial^{\mathcal{I}(a),s_k}_{\mathcal{I}(b)} - \\ - \sum_{x \in \mathcal{I}(b) \setminus \mathcal{I}(d)} Partial^{s_k}_{\mathcal{I}(a)}(x) + \sum_{x \in \mathcal{I}(d) \setminus \mathcal{I}(b)} Partial^{s_k}_{\mathcal{I}(a)}(x), \quad \forall a \in \mathcal{V}$$

with $|\mathcal{I}(b) \ominus \mathcal{I}(d)|$ operations being performed.

(ii) $s_{k+1}(a,d)$, $\forall a \in \mathcal{V}\backslash\{d\}$, can be computed as

$$s_{k+1}(a,d) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(d)|} OuterPartial^{\mathcal{I}(a),s_k}_{\mathcal{I}(d)}, \quad \forall a \in \mathcal{V}\backslash\{d\}. \tag{2.10}$$

(The proof is similar to Proposition 2.6. We omit it here.)

We next provide an example to illustrate how to use outer partial sums sharing for further speeding up the computation of SimRank.

Example 2.9 (Use outer partial sums sharing for speeding up SimRank computation)**.** Recall the graph $\mathcal{G}$ in Figure 2.1a, with the (inner) partial sums sharing dendrogram in Figure 2.3b. Suppose $Partial^{s_k}_{\mathcal{I}(x)}(\star)$, $\forall x \in \mathcal{V}$, have been pre-computed via Example 2.7, as depicted in part in the first four columns of Figure 2.4. We show how to compute $s_{k+1}(x,a)$ and $s_{k+1}(x,c)$, $\forall x \in \mathcal{V}$, by using outer partial sums sharing.

Firstly, for each non-empty in-neighbor set $\mathcal{I}(x)$, we compute $OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(a)}$ and $OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(c)}$, $\forall x \in \mathcal{V}$, from the cached results of $Partial^{s_k}_{\mathcal{I}(x)}(\star)$. In light

of the clustering dendrogram in Figure 2.3b, we notice that the item '$\boxed{b} \oplus \boxed{g} \cdots \mathcal{I}(a)$', which, in the context of *outer* partial sums, can be reinterpreted as "adding up the (inner) partial sums $Partial^{s_k}_{\mathcal{I}(x)}(b)$ and $Partial^{s_k}_{\mathcal{I}(x)}(g)$ to yield the outer partial sums $OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(a)}$, for all $x \in \mathcal{V}$". Thus, we have

$$OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(a)} = \sum_{y \in \{b,g\}} Partial^{s_k}_{\mathcal{I}(x)}(y). \quad (\forall x \in \mathcal{V})$$

For instance, $OuterPartial^{\mathcal{I}(b),s_k}_{\mathcal{I}(a)} = 0.15 + 1 = 1.15$, for $x = b$, as illustrated in row 'b' of Figure 2.4.

Similarly, the item '$\mathcal{I}(a) \oplus \boxed{d} \cdots \mathcal{I}(c)$' in Figure 2.3b implies that $OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(c)}$, $\forall x \in \mathcal{V}$, can be calculated from the cached results of $OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(a)}$ via Eq.(2.9) as

$$\begin{aligned} OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(c)} &= OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(a)} \\ &\quad + Partial^{s_k}_{\mathcal{I}(x)}(d), \quad (\forall x \in \mathcal{V}) \end{aligned}$$

e.g., $OuterPartial^{\mathcal{I}(b),s_k}_{\mathcal{I}(c)} = 1.15 + 0.08 = 1.23$, for $x = b$.

The rest of the results are shown in Cols 5-6 of Figure 2.4.

Then, using Eq.(2.10), we can obtain $s_{k+1}(x, a)$ and $s_{k+1}(x, c)$, $\forall x \in \mathcal{V}$, from the memoized results of $OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(a)}$ and $OuterPartial^{\mathcal{I}(x),s_k}_{\mathcal{I}(c)}$. For example, in row 'b' of Figure 2.4,

$$s_{k+1}(b, a) = \frac{0.6}{2 \times 4} \times 1.15 = 0.09, \quad (x = b)$$

$$s_{k+1}(b, c) = \frac{0.6}{3 \times 4} \times 1.23 = 0.06. \quad (x = b)$$

The remainder of the similarities are depicted in the last two columns of Figure 2.4. □

2.3.3 A SimRank Algorithm

We next present a complete algorithm to efficiently compute SimRank, by integrating the aforementioned techniques of inner and outer partial sums sharing.

The main result of this subsection is the following.

Algorithm 2.1: OIP-SR $(\mathcal{G}, C, K)$

Input : graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, damping factor C,
iteration number K.

Output: SimRank scores $s_K(\star, \star)$.

construct a transitional MST $\mathscr{T} \leftarrow$ DMST-Reduce $(\mathcal{G})$;

initialize $s_0(x, y) \leftarrow \begin{cases} 1, & x=y \\ 0, & x \neq y \end{cases} \quad \forall x, y \in \mathcal{V}$

for $k \leftarrow 0, 1, \cdots, K-1$ **do**

  foreach *vertex* $u \in \mathcal{O}(\#)$ *in the MST* $\mathscr{T}$ **do**

    foreach *vertex* $y \in \mathcal{V}$ *in* $\mathcal{G}$ **do**

      $Partial^{s_k}_{\mathcal{I}(u)}(y) \leftarrow \sum_{x \in \mathcal{I}(u)} s_k(x, y)$;

    $s_{k+1}(u, \star) \leftarrow$ OP $(\mathscr{T}, \mathcal{G}, u, C, k, Partial^{s_k}_{\mathcal{I}(u)}(\star))$;

    while $\mathcal{O}(u) \neq \varnothing$ **do**

      $v \leftarrow \mathcal{O}(u)$;

      foreach *vertex* $y \in \mathcal{V}$ *in* $\mathcal{G}$ **do**

        $$Partial^{s_k}_{\mathcal{I}(v)}(y) \leftarrow Partial^{s_k}_{\mathcal{I}(u)}(y) - \sum_{x \in \mathcal{I}(u) \setminus \mathcal{I}(v)} s_k(x, y) + \sum_{x \in \mathcal{I}(v) \setminus \mathcal{I}(u)} s_k(x, y);$$

      $s_{k+1}(v, \star) \leftarrow \mathsf{OP}(\mathscr{T}, \mathcal{G}, v, C, k, Partial^{s_k}_{\mathcal{I}(v)}(\star))$;

      $u \leftarrow v$;

    foreach *vertex* $y \in \mathcal{V}$ *in* $\mathcal{G}$ **do**

      free $Partial^{s_k}_{\mathcal{I}(u)}(y)$;

      while $\mathcal{O}(u) \neq \varnothing$ **do**

        $v \leftarrow \mathcal{O}(u)$, free $Partial^{s_k}_{\mathcal{I}(v)}(y)$, $u \leftarrow v$;

return $s_K(\star, \star)$;

Proposition 2.10. For any graph $\mathcal{G}$, it is in $O(dn^2 + Kd'n^2)$ time and $O(n)$ intermediate memory to compute SimRank similarities of all pairs of vertices for K iterations, where d is the average vertex in-degree of $\mathcal{G}$, and $d' \leq d$.

Note that d' is affected by the overlapped area size among different in-neighbor sets in $\mathcal{G}$. Typically, d' is much smaller than d as in-neighbor sets in $\mathcal{G}$ may have many vertices in common in real networks. That is, our approach of partial sums sharing can compute SimRank more efficiently than psum-SR in practice, as opposed to the $O(Kdn^2)$-time of the conventional counterpart via separate partial sums over each in-neighbour set in $\mathcal{G}$. Even in the extreme case when all in-neighbor sets in $\mathcal{G}$ are pair-wise disjoint, our method can retain the same complexity bound of psum-SR in the worst case.

We next prove Proposition 2.10 by providing an algorithm for SimRank computation, with the desired complexity bound.

Algorithm. The algorithm, referred to as OIP-SR, is shown in Algorithm 2.1. Given $\mathcal{G}$, a damping factor C, and the total iteration number K, it returns $s_K(\star,\star)$ of all pairs of vertices.

In the sequel, we shall abuse the notation $\mathcal{O}(v)$ to denote the out-neighbor set of vertex v.

The algorithm OIP-SR works as follows. (1) It first invokes procedure DMST-Reduce to identify the topological sort based on a minimum spanning tree $\mathscr{T}$ for computing partial sums (line 1). (2) For each iteration k, OIP-SR checks each path in $\mathscr{T}$, starting from the root node # as follows. (a) For the first edge $(\#, u)$ in each path, OIP-SR computes $Partial^{s_k}_{\mathcal{I}(u)}(\star)$ from scratch (lines 5-6), and then invokes procedure OP to compute $s_{k+1}(u,\star)$ by outer partial sums sharing (line 7). (b) For other edges (u, v) in each path, OIP-SR computes $Partial^{s_k}_{\mathcal{I}(v)}(\star)$ from the result of $Partial^{s_k}_{\mathcal{I}(u)}(\star)$ memoized earlier (lines 10-11), and gets $s_{k+1}(v,\star)$ by invoking procedure OP of outer partial sums sharing (line 12). This process repeats until all edges in every path have been traversed, and OIP-SR frees the memoized results of the partial sums generated from each path (lines 14-17). (3) The loop will continue to iterate until k reaches K, and OIP-SR returns all the similarities $s_K(\star,\star)$ (line 18).

The procedures of OP and DMST-Reduce are described below.

Procedure DMST-Reduce($\mathcal{G}$)

Input : graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

Output: transitional MST $\mathscr{T}$.

1 initialize $\mathscr{V} \leftarrow \mathcal{V} \cup \{\#\}$, $\mathscr{E} \leftarrow \varnothing$;

2 sort the vertices of $\mathcal{G}$ into non-decreasing order by in-degree ;

3 initialize $\mathcal{U} \leftarrow \mathcal{V}$;

4 **foreach** *vertex* $a \in \mathcal{V}$ *in* $\mathcal{G}$, *taken in sorted order* **do**

5 $\quad \mathcal{U} \leftarrow \mathcal{U} \backslash \{a\}$;

6 $\quad$ **foreach** *vertex* $b \in \mathcal{U}$ *in* $\mathcal{G}$, *taken in sorted order* **do**

7 $\quad\quad \mathscr{E} \leftarrow \mathscr{E} \cup \{(a, b)\}$;

8 $\quad\quad$ assign a weight w to the edge (a, b) of $\mathscr{E}$:

$\quad\quad\quad w(a, b) \leftarrow \min\{|\mathcal{I}(a) \ominus \mathcal{I}(b)|, |\mathcal{I}(b)| - 1\}$;

9 find the MST $\mathscr{T}$ of the graph $\mathscr{G} = (\mathscr{V}, \mathscr{E}, w)$:

$\quad \mathscr{T} \leftarrow$ Directed-MST $(\mathscr{G}, \#, w)$;

10 **return** $\mathscr{T}$;

Procedure DMST-Reduce. Given a graph $\mathcal{G}$, the procedure returns a minimum spanning tree $\mathscr{T}$ as a topological sort for computing partial sums. First, it builds a weighed graph $\mathscr{G}$, whose edge weights are the transition costs of all pairs of vertices (plus a special # denoting 'the root node') in $\mathcal{G}$ (lines 1-8). Then, it runs an algorithm [GGST86] to find a directed MST $\mathscr{T}$ of $\mathscr{G}$ (starting from vertex #), which is returned as the final result (lines 9-10).

Procedure OP. This procedure adopts a similar paradigm of OIP-SR for outer partial sums sharing. The procedure OP takes as input a topological sort $\mathscr{T}$, a graph $\mathcal{G}$, a vertex u, a damping factor C, iteration k, and the cached partial sums $Partial^{s_k}_{\mathcal{I}(u)}(\star)$. It returns the similarities $s_{k+1}(u, \star)$.

The procedure OP runs in three phases for each path that starts from the root # of

Procedure $\mathsf{OP}(\mathscr{T}, \mathcal{G}, u, C, k, Partial^{s_k}_{\mathcal{I}(u)}(\star))$

Input : transitional MST $\mathscr{T}$, graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$,
vertex u, damping factor C,
iteration k, partial sums $Partial^{s_k}_{\mathcal{I}(u)}(\star)$.

Output: SimRank scores $s_{k+1}(u, \star)$.

1 **foreach** *vertex* $w \in \mathcal{O}(\#)$ *in the MST* $\mathscr{T}$ **do**
2 $\quad OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(w)} \leftarrow \sum_{y \in \mathcal{I}(w)} Partial^{s_k}_{\mathcal{I}(u)}(y)$;
3 $\quad$ **if** $u = w$ **then** $s_{k+1}(u, w) \leftarrow 1$;
4 $\quad$ **else if** $\mathcal{I}(u) = \varnothing$ *or* $\mathcal{I}(w) = \varnothing$ **then** $s_{k+1}(u, w) \leftarrow 0$;
5 $\quad$ **else** $s_{k+1}(u, w) \leftarrow \frac{C}{|\mathcal{I}(u)||\mathcal{I}(w)|} OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(w)}$;
6 $\quad$ **while** $\mathcal{O}(w) \neq \varnothing$ **do**
7 $\quad\quad z \leftarrow \mathcal{O}(w)$;
8 $\quad\quad OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(z)} \leftarrow OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(w)} - \sum_{y \in \mathcal{I}(w) \setminus \mathcal{I}(z)} Partial^{s_k}_{\mathcal{I}(u)}(y) + \sum_{y \in \mathcal{I}(z) \setminus \mathcal{I}(w)} Partial^{s_k}_{\mathcal{I}(u)}(y)$;
9 $\quad\quad$ **if** $u = z$ **then** $s_{k+1}(u, z) \leftarrow 1$;
10 $\quad\quad$ **else if** $\mathcal{I}(u) = \varnothing$ *or* $\mathcal{I}(z) = \varnothing$ **then** $s_{k+1}(u, z) \leftarrow 0$;
11 $\quad\quad$ **else** $s_{k+1}(u, z) \leftarrow \frac{C}{|\mathcal{I}(u)||\mathcal{I}(z)|} OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(z)}$;
12 $\quad\quad w \leftarrow z$;
13 $\quad$ free $OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(w)}$;
14 $\quad$ **while** $\mathcal{O}(w) \neq \varnothing$ **do**
15 $\quad\quad z \leftarrow O(w)$, free $OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(z)}$, $w \leftarrow z$;
16 **return** $s_{k+1}(u, \star)$;

the tree $\mathscr{T}$. (a) For the first edge $(\#, w)$ of each path, OP needs to start from scratch to calculate $OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(w)}$ (line 2) and $s_{k+1}(u, w)$ (lines 3-5) from the memoized $Partial^{s_k}_{\mathcal{I}(u)}(\star)$. (b) For other edges (w, z) in each path, OP obtains $OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(z)}$ from the cached result of $OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(w)}$ (line 8), and then computes $s_{k+1}(u, z)$ (lines

9-11). The loop continues until all edges in the path have been visited. (c) OP releases the memoized results of all the outer partial sums which are generated by each path (lines 13-15). The whole process repeats until all the paths in $\mathscr{T}$ have been processed, and returns $s_{k+1}(u,\star)$ (line 16).

Correctness & Complexity. OIP-SR consists of two phases: (i) building an MST $\mathscr{T}$ (line 1), and (ii) computing similarities (lines 2-18). One can readily verify that (1) OIP-SR correctly computes the similarities $s_k(u,v)$ in $\mathcal{G}$ for each vertex pair (u,v); and (2) the total time of OIP-SR is bounded by $O(Kd'n^2)$, with $d' \leq d$.

(1) Correctness. (i) Algorithm OIP-SR correctly computes the similarities $s_k(u,v)$ in $\mathcal{G}$ for each vertex pair (u,v). One can verify that after the **foreach** loops (lines 5-6 and lines 10-11), for every vertex $u \in \mathscr{T}$, $Partial^{s_k}_{\mathcal{I}(u)}(\star)$ and $OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(\star)}$ are memoized, and the similarities $s_{k+1}(u,\star)$ are computed. (ii) The partial sums computed by our algorithm are indeed *optimized* because while computing $Partial^{s_k}_{\mathcal{I}(u)}(\star)$ and $OuterPartial^{\mathcal{I}(u),s_k}_{\mathcal{I}(\star)}$ for each vertex u, we allow the common parts of partial sums to be recomputed as fewer as possible by virtue of a minimum spanning tree $\mathscr{T}$; in particular, the partial sums sharing would definitely happen in every path of $\mathscr{T}$ for a graph with $|\bigcup_{v\in\mathcal{V}} \mathcal{I}(v)|$ less than $\sum_{v\in\mathcal{V}} |\mathcal{I}(v)|$.

(2) Complexity. OIP-SR consists of two phases: (i) building an MST $\mathscr{T}$ (line 1), and (ii) computing similarities (lines 2-18). We analyze the time for each phase below.

(i) The procedure DMST-Reduce is used for finding a directed MST $\mathscr{T}$, which is bounded by $O(dn^2)$ time and $O(n)$ space. It includes (a) $O(n \log n)$ time and $O(n)$ space for sorting vertices in $\mathcal{G}$ by in-degree (line 2), (b) $O(d)$ time and $O(2d)$ space for computing the transitional cost for a single edge (a,b) in $\mathscr{E}$, being $O(\frac{dn^2}{2})$ time for all edges in $\mathscr{E}$ (lines 4-8), and (c) $O(n^2 \log n)$ time and $O(n)$ space for finding the MST $\mathscr{T}$ of $\mathcal{G}$ [GGST86].

(ii) For each iteration, OIP-SR uses $\mathscr{T}$ rooted at # to compute similarities in $\mathcal{G}$. Note that $|\mathcal{O}(\#)|$ paths in $\mathscr{T}$ are used for calculating partial sums over all in-neighbour sets

of $\mathcal{G}$. Therefore, for completing a single path of average length $\frac{n}{|\mathcal{O}(\#)|}$, the complexity required for computing the partial sums, for the first edge of the path, is $O(nd)$ time and $O(n)$ space (lines 5-6); the complexity required, apart from the first edge of the path, is $O(\frac{n}{|\mathcal{O}(\#)|} \cdot n \cdot d_{\ominus})$ time and $O(n)$ space, with $d_{\ominus} \triangleq \text{avg}_{(u,v)\in\mathscr{T}}|\mathcal{I}(u) \ominus \mathcal{I}(v)|$ (lines 8-13). It follows that the total complexity bound in this phase is $O(K(|\mathcal{O}(\#)| \cdot nd + n^2 \cdot d_{\ominus}))$ time and $O(n)$ space for K iterations. Since $d_{\ominus} \ll d$ and $|\mathcal{O}(\#)| \ll n$, such a time complexity bound is far less than $O(Kdn^2)$.

Combining (i) and (ii), the total complexity of OIP-SR is $O(dn^2 + K(|\mathcal{O}(\#)| \cdot nd + n^2 \cdot d_{\ominus}))$ time and $O(n)$ space.

2.4 Exponential Rate of Convergence

For a desired accuracy ϵ, the existing paradigm (via Eq.(2.1)) for computing SimRank needs $K = \lceil \log_C \epsilon \rceil$ iterations [LVGT10]. In this section, we introduce a new notion of SimRank that is based on a matrix differential equation, which can significantly reduce the number of iterations for attaining the accuracy ϵ while fairly preserving the relative order of SimRank.

The main idea in our approach is to replace the geometric sum of the conventional SimRank by an exponential sum that provides more rapid rate of convergence. We start by expanding the conventional SimRank matrix form (in Eq.(2.2))

$$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n,$$

as a power series:

$$\mathbf{S} = (1 - C) \cdot \sum_{i=0}^{\infty} C^i \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i, \tag{2.11}$$

where we notice that the coefficient for each term in the summation makes a geometric sequence $\{1, C, C^2, \cdots\}$. For this expansion form, the effect of damping factor C^i in the summation is to reduce the contribution of long paths relative to short ones. That is, the conventional SimRank measure considers two vertices to be more similar if they have more paths of short length between them. Following this intuition, we observe that there

is an opportunity to speed up the asymptotic rate of convergence for SimRank iterations, if we allow a slight (and with hindsight sensible) modification of Eq.(2.11) as follows:

$$\hat{\mathbf{S}} = e^{-C} \cdot \sum_{i=0}^{\infty} \frac{C^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i, \tag{2.12}$$

Comparing Eq.(2.11) with Eq.(2.12), we notice that $\hat{\mathbf{S}}$ is just an exponential sum rather than $\mathbf{S}$ that is a geometric sum. Since the exponential sum converges more rapidly, such a modification can speed up the computation of SimRank. In addition, the modified coefficient for each term in the summation of Eq.(2.12) that yields the exponential sequence $\{1, \frac{C}{1!}, \frac{C^2}{2!}, \cdots\}$ still obeys the intuition of the conventional counterpart, *i.e.,* the efficacy of damping factor $\frac{C^i}{i!}$ is to reduce the contribution of long paths relative to short ones.

2.4.1 Closed Form of Exponential SimRank

With the modified notion of SimRank in Eq.(2.12), we now need to define an Eq.(2.2)-like recurrence for $\hat{\mathbf{S}}$.

Definition 2.11. Let $\hat{\mathbf{S}}(t)$ be a matrix function *w.r.t.* a scalar t. The *matrix differential form of SimRank* is defined to be $\hat{\mathbf{S}} \triangleq \hat{\mathbf{S}}(t)|_{t=C}$ such that $\hat{\mathbf{S}}(t)$ satisfies the following matrix differential equation:

$$\frac{\mathrm{d}\hat{\mathbf{S}}(t)}{\mathrm{d}t} = \mathbf{Q} \cdot \hat{\mathbf{S}}(t) \cdot \mathbf{Q}^T, \qquad \hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n. \qquad \square \tag{2.13}$$

Note that the solution of Eq.(2.13) is unique since the initial condition $\hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n$ is specified. Based on Definition 2.11, it is crucial to verify that $\hat{\mathbf{S}}$ (in Eq.(2.12)) is the solution to Eq.(2.13). Proposition 2.12 shows the correctness.

Proposition 2.12. The matrix differential form of SimRank in Eq.(2.13) has an exact solution $\hat{\mathbf{S}}$ given in Eq.(2.12). $\square$

Proof. We shall prove this by plugging

$$\hat{\mathbf{S}}(t) = A \cdot (\mathbf{I}_n + \sum_{i=1}^{\infty} \frac{t^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i),$$

with an arbitrary constant A, into the SimRank differential formula Eq.(2.13):

$$\begin{aligned}\frac{\mathrm{d}\hat{\mathbf{S}}(t)}{\mathrm{d}t} &= A \cdot \sum_{i=1}^{\infty} \frac{\mathrm{d}}{\mathrm{d}t}\left(\frac{t^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i\right) \\ &= A \cdot \sum_{i=1}^{\infty} \frac{t^{i-1}}{(i-1)!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i = \mathbf{Q} \cdot \hat{\mathbf{S}}(t) \cdot \mathbf{Q}^T,\end{aligned}$$

where the first equality holds because we notice that $\left\|\frac{t^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i\right\|_{\max} \leq \frac{t^i}{i!}$, and the series $\sum_{i=1}^{\infty} \frac{t^i}{i!}$ converges uniformly on $t \in [0, C]$.

Thus, we have verified that the solution to Eq.(2.13) takes the form

$$\hat{\mathbf{S}}(t) = A \cdot \left(\mathbf{I}_n + \sum_{i=1}^{\infty} \frac{t^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i\right).$$

To find A, let $t = 0$ and $\hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n$. Then we have $A \cdot \mathbf{I}_n = e^{-C} \cdot \mathbf{I}_n$, which implies that $A = e^{-C}$. Therefore,

$$\hat{\mathbf{S}}(t) = e^{-C} \cdot \sum_{i=0}^{\infty} \frac{t^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i.$$

Setting $t = C$, we obtain $\hat{\mathbf{S}} \triangleq \hat{\mathbf{S}}(C)$, the solution to Eq.(2.13). □

To iteratively compute $\hat{\mathbf{S}}$, the conventional way is to use *the Euler method* [AP98] for approximating $\hat{\mathbf{S}}(t)$ at time $t = C$. Precisely, by choosing a value h for the step size, and setting $t_k = k \cdot h$, one step of the Euler method from t_k to t_{k+1} is

$$\hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + h \cdot \mathbf{Q} \cdot \hat{\mathbf{S}}_k \cdot \mathbf{Q}^T, \quad \hat{\mathbf{S}}_0 = \hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n.$$

Subsequently, the value of $\hat{\mathbf{S}}_k$ is an approximation of the solution to Eq.(2.13) at time $t = t_k$, *i.e.*, $\hat{\mathbf{S}}_k \approx \hat{\mathbf{S}}(t_k)$. However, the approximation error of the Euler method hinges heavily on the choice of step size h, which is hard to determine since the small choice of h would entail huge computational cost for attaining high accuracy. To address this issue, we adopt the following iterative paradigm for computing $\hat{\mathbf{S}}$ as

$$\begin{cases} \mathbf{T}_{k+1} = \mathbf{Q} \cdot \mathbf{T}_k \cdot \mathbf{Q}^T \\ \hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + e^{-C} \cdot \frac{C^{k+1}}{(k+1)!} \cdot \mathbf{T}_{k+1} \end{cases} \text{with} \begin{cases} \mathbf{T}_0 = \mathbf{I}_n \\ \hat{\mathbf{S}}_0 = e^{-C} \cdot \mathbf{I}_n \end{cases} \tag{2.14}$$

Note that the main difference in our approach, as compared to the Euler method, is that there is no need for the choice of a particular step size h to iteratively compute $\hat{\mathbf{S}}$. The correctness of our approach can be easily verified, by induction on k, that the value of $\hat{\mathbf{S}}_k$ in our iteration Eq.(2.14) equals the sum of the first k terms of the infinite series $\hat{\mathbf{S}}$ in Eq.(2.12).

2.4.2 A Space-Efficient Iterative Paradigm

Although the paradigm of Eq.(2.14) can iteratively compute $\hat{\mathbf{S}}_k$ that converges to the exponential SimRank $\hat{\mathbf{S}}$, we observe that Eq.(2.14) requires additional memory space to store the intermediate result $\mathbf{T}_k$ per iteration. In this subsection, we provide an improved version of Eq.(2.14) that can produce the same result without using extra space for caching $\mathbf{T}_k$.

Proposition 2.13. Given any total iteration number K, the following paradigm can be used to iteratively compute $\tilde{\mathbf{S}}_K$:

$$\begin{cases} \tilde{\mathbf{S}}_0 = e^{-C} \cdot \mathbf{I}_n, \\ \tilde{\mathbf{S}}_{k+1} = \frac{C}{K-k} \cdot \mathbf{Q} \cdot \tilde{\mathbf{S}}_k \cdot \mathbf{Q}^T + e^{-C} \cdot \mathbf{I}_n. \quad (k = 0, \cdots, K-1) \end{cases} \tag{2.15}$$

The result of $\tilde{\mathbf{S}}_K$ at the last iteration is exactly the same as $\hat{\mathbf{S}}_K$ in Eq.(2.14). □

The main idea of our improved paradigm Eq.(2.15) is based on two observations: (1) For every iteration $k = 0, 1, \cdots, K$, the result of $\hat{\mathbf{S}}_k$ in Eq.(2.14) is actually the sum of the first k terms of the infinite series $\hat{\mathbf{S}}$ in Eq.(2.12). (2) For any total iteration number K, the result of $\tilde{\mathbf{S}}_K$ at the last iteration in Eq.(2.15) equals the sum of the first K terms of the infinite series $\hat{\mathbf{S}}$ in Eq.(2.12). Both of these observations can be readily verified by direct inductive manipulations. As an example for $K = 3$, our improved paradigm Eq.(2.15) iteratively computes $\hat{\mathbf{S}}_3 = e^{-C} \cdot \sum_{i=0}^{3} \frac{C^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i$ as follows:

$$\tilde{\mathbf{S}}_3 = e^{-C}\mathbf{I}_n + C\mathbf{Q}\underbrace{\left(e^{-C}\mathbf{I}_n + \frac{C}{2}\mathbf{Q}\overbrace{\left(e^{-C}\mathbf{I}_n + \frac{C}{3}\mathbf{Q}\cdot\mathbf{Q}^T\right)}^{\tilde{\mathbf{S}}_1}\mathbf{Q}^T\right)}_{\tilde{\mathbf{S}}_2}\mathbf{Q}^T.$$

The merit of Eq.(2.15) over Eq.(2.14) is the space efficiency — in Eq.(2.15), we do not need to use an auxiliary matrix $\mathbf{T}_k$ to store the temporary results. Moreover, since Eq.(2.15) has a very similar form to the SimRank matrix form in Eq.(2.2), our partial sums sharing techniques in Section 2.3 can be directly applied to the iterative form of Eq.(2.15), *i.e.,* when $a \neq b$, for $k = 0, 1, \cdots, K-1$,

$$[\tilde{\mathbf{S}}_{k+1}]_{a,b} = \frac{C}{(K-k)|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} [\tilde{\mathbf{S}}_k]_{i,j}.$$

It is worth noticing that in Eq.(2.14), we can iteratively compute $\hat{\mathbf{S}}_{k+1}$ from $\hat{\mathbf{S}}_k$ for *any* $k = 0, 1, \cdots$, whereas, in Eq.(2.15), for any given K, we can only iteratively compute $\tilde{\mathbf{S}}_{k+1}$ from $\tilde{\mathbf{S}}_k$ for $k = 0, 1, \cdots, K-1$, but we cannot compute $\tilde{\mathbf{S}}_{K+1}$ from $\tilde{\mathbf{S}}_K$. This means that, to guarantee a given accuracy ϵ, we have to determine the total number of iterations K in an *a-priori* fashion for Eq.(2.15), in contrast with Eq.(2.14) in which K can be determined in an either *a-priori* or *a-posteriori* style. Fortunately, this requirement is not an obstacle to Eq.(2.15), since in the next subsection we will show a nice *a-priori* bound of the total iteration number K for Eq.(2.15) to attain a given accuracy ϵ.

2.4.3 Error Estimate

In the SimRank matrix differential model, the following estimate for the k-th iterative similarity matrix $\hat{\mathbf{S}}_k$ with respect to the exact one $\hat{\mathbf{S}}$ can be established.

Proposition 2.14. For each iteration $k = 0, 1, 2, \cdots$, the difference between the k-th iterative and the exact similarity matrix in Eqs.(2.12) and (2.14) can be bounded as follows:

$$\|\hat{\mathbf{S}}_k - \hat{\mathbf{S}}\|_{\max} \leq \frac{C^{k+1}}{(k+1)!}, \tag{2.16}$$

where $\|\mathbf{X}\|_{\max} \triangleq \max_{i,j} |x_{i,j}|$ is the max norm. □

Proof. Subtracting Eq.(2.12) from Eq.(2.14), we obtain

$$\hat{\mathbf{S}}_k - \hat{\mathbf{S}} = e^{-C} \cdot \sum_{i=k+1}^{\infty} \frac{C^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i.$$

Taking the matrix-to-vector operator $vec(\star)$ [LHH+10] on both sides, and then applying the Kronecker product property that $vec(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A}) \cdot vec(\mathbf{X})$ to the right-hand side gives

$$vec(\hat{\mathbf{S}}_k - \hat{\mathbf{S}}) = e^{-C} \cdot \sum_{i=k+1}^{\infty} \frac{C^i}{i!} \cdot (\mathbf{Q} \otimes \mathbf{Q})^i \cdot vec(\mathbf{I}_n),$$

Notice that $\mathbf{Q}$ is a transitional matrix, *i.e.,* the sum of each row in $\mathbf{Q}$ is less than 1, which implies that $\|\mathbf{Q} \otimes \mathbf{Q}\|_\infty \leq 1$.

Take the matrix ∞-norm $\| \star \|_\infty$ on both sides, and apply $\|vec(\star)\|_\infty = \| \star \|_{\max}$ to the left-hand side:

$$\begin{aligned} \|\hat{\mathbf{S}}_k - \hat{\mathbf{S}}\|_{\max} &\leq e^{-C} \cdot \sum_{i=k+1}^{\infty} \frac{C^i}{i!} \cdot \|(\mathbf{Q} \otimes \mathbf{Q})\|_\infty^i \cdot \|vec(\mathbf{I}_n)\|_\infty \\ &\leq e^{-C} \cdot \sum_{i=k+1}^{\infty} \frac{C^i}{i!} \leq \frac{C^{k+1}}{(k+1)!}, \end{aligned}$$

where the last inequality holds because using the Lagrange remainder $\frac{f^{(k+1)}(\xi)}{(k+1)!}C^{k+1}$, $\xi \in (0, C)$, of Maclaurin series for $f(C) = e^C$ yields

$$\sum_{i=k+1}^{\infty} \frac{C^i}{i!} = \frac{e^\xi}{(k+1)!}C^{k+1} \leq \frac{e^C}{(k+1)!}C^{k+1},$$

which completes the proof. □

For the SimRank differential model Eq.(2.12), Proposition 2.14 allows finding out the exact number of iterations needed for attaining a desired accuracy, based on the following corollary.

Corollary 2.15. For a desired accuracy $\epsilon > 0$, the number of iterations required to perform Eq.(2.14) is

$$K' \geq \lceil \frac{\ln \epsilon'}{W(\frac{1}{e \cdot C} \cdot \ln \epsilon')} \rceil, \text{ with } \epsilon' = (\sqrt{2\pi} \cdot \epsilon)^{-1}.$$

Here, $W(\star)$ is the Lambert W function [Has05]. □

Proof. Based on Eq.(2.16), $\forall \epsilon > 0$, we need to find an integer $K' > 0$ such that $\frac{C^{K'+1}}{(K'+1)!} \leq \epsilon$.

We first use the Stirling's formula

$$(K'+1)! \geq \sqrt{2\pi} \cdot (\frac{K'+1}{e})^{K'+1}$$

to obtain $(\frac{e \cdot C}{K'+1})^{K'+1} \leq \sqrt{2\pi} \cdot \epsilon$.

Let $x = \frac{K'+1}{e \cdot C}$. It follows that $x^x \geq (\sqrt{2\pi} \cdot \epsilon)^{-\frac{1}{e \cdot C}}$. Using the Lambert W function, we have

$$x \geq \frac{\ln (\sqrt{2\pi} \cdot \epsilon)^{-\frac{1}{e \cdot C}}}{W(\ln (\sqrt{2\pi} \cdot \epsilon)^{-\frac{1}{e \cdot C}})}.$$

By substituting $x = \frac{K'+1}{e \cdot C}$ back into the inequality, we get the final result. □

Noting that $\ln(x) - \ln(\ln(x)) \leq W(x) \leq \ln(x)$, $\forall x > e$ [Has05], we have the following improved version of Corollary 2.15, which may avoid computing the Lambert W function.

Corollary 2.16. For a desired accuracy $0 < \epsilon < \frac{1}{\sqrt{2\pi}} e^{-C \cdot e^2}$, the number of iterations needed to perform Eq.(2.14) is

$$K' \geq \lceil tfrac - \ln(\sqrt{2\pi} \cdot \epsilon)\eta - \ln(\eta) \rceil \text{ with } \eta = \ln(-\tfrac{1}{e \cdot C} \cdot \ln(\sqrt{2\pi} \cdot \epsilon)). \quad \square$$

Comparing this with the conventional SimRank model that requires $K = \lceil \log_C \epsilon \rceil$ iterations [LVGT10] for a given accuracy ϵ, we see that our revision of the differential SimRank model in Eq.(2.13) can greatly speed up the convergence of SimRank iterations from the original geometric to exponential rate.

As an example, setting $C = 0.8$ and $\epsilon = 0.0001$, since $\frac{1}{\sqrt{2\pi}} e^{-0.8 \cdot e^2} = 0.0011 > 0.0001$, we can use Corollary 2.16 to find out the number of iterations K' in Eq.(2.14) necessary to our differential SimRank model Eq.(2.13) as follows:

$$\eta = \ln(-\frac{1}{e \cdot 0.8} \cdot \ln(\sqrt{2\pi} \cdot 0.0001)) = 1.3384,$$

$$K' \geq \lceil \frac{-\ln(\sqrt{2\pi} \cdot 0.0001)}{1.3384 - \ln(1.3384)} \rceil = \lceil \frac{8.2914}{1.0469} \rceil = 7.$$

In contrast, the conventional SimRank model Eq.(2.1) needs $K = \lceil \log_{0.8} 0.0001 \rceil = 41$ iterations.

For ranking purpose, our experimental results in Section 2.6 further show that the revised notion of SimRank in Eq.(2.13) not only drastically reduces the number of iterations for a desired accuracy, but can fairly maintain the relative order of vertices with respect to the conventional SimRank in [LVGT10].

2.5 Partial Max Sharing for Minimax SimRank Variation in Bipartite Graphs

Having investigated the partial *sums* sharing problem for optimizing SimRank computation in Section 2.4, we now focus on the partial *max* sharing problem for optimizing the computation of *the Minimax SimRank variation*, a model proposed in [JW02, Section 4.3.2] (see Definition 1.9).

To compute $s(A, B)$ for bipartite SimRank, the conventional method is to perform the following iterations:

$$s_0(A,B) \;=\; \begin{cases} 1, & A = B; \\ 0, & A \neq B. \end{cases}$$

For $k \geq 0$, we define (i) $s^A_{k+1}(A,B) = 0$ if $\mathcal{O}(A) = \varnothing$; (ii) $s^B_{k+1}(A,B) = 0$ if $\mathcal{O}(B) = \varnothing$; (iii) otherwise,

$$s^A_{k+1}(A,B) \;=\; \frac{C}{|\mathcal{O}(A)|} \sum_{i \in \mathcal{O}(A)} \max_{j \in \mathcal{O}(B)} s_k(i,j), \tag{2.17}$$

$$s^B_{k+1}(A,B) \;=\; \frac{C}{|\mathcal{O}(B)|} \sum_{j \in \mathcal{O}(B)} \max_{i \in \mathcal{O}(A)} s_k(i,j), \tag{2.18}$$

$$s_{k+1}(A,B) \;=\; \min\{s^A_{k+1}(A,B), s^B_{k+1}(A,B)\}. \tag{2.19}$$

We can readily prove that

$$\lim_{k \to \infty} s_k(A,B) = s(A,B).$$

To speed up the computation of $s_k(\star,\star)$ for all pairs of vertices, we can first memoize

the partial max in Eq.(2.17) [5] as follows:

$$Partial\_Max^{s_k}_{\mathcal{O}(B)}(i) = \max_{j \in \mathcal{O}(B)} s_k(i,j), \tag{2.20}$$

and then compute $s^A_{k+1}(A,B)$ as

$$s^A_{k+1}(A,B) = \frac{C}{|\mathcal{O}(A)|} \sum_{i \in \mathcal{O}(A)} Partial\_Max^{s_k}_{\mathcal{O}(B)}(i). \tag{2.21}$$

Thus, the memoized results of $Partial\_Max^{s_k}_{\mathcal{O}(B)}(\star)$ can be reused in all $s^X_{k+1}(X,B)$ computations, $\forall X \in \mathcal{V}$.

It should be pointed out that, although Eqs.(2.20) and (2.21) have a very similar form to Eqs.(2.3) and (2.4), we only can apply the (outer) partial sums sharing technique of Section 2.3.2 to further speed up the summations in Eq.(2.21), but may not always employ the (inner) partial sums sharing technique of Section 2.3.1 to accelerate the partial max computation in Eq.(2.20). The reason is that, for partial *sums* sharing, "subtraction" is allowed to compute one partial sum from another (see Eq.(2.8) in Proposition 2.6), whereas, for partial *max* sharing, "subtraction" is disallowed in the context of "max" operator since *the maximum value of a set X may be unequal to the maximum value of a subset of X*. We call this *the "subtraction" curse* of max operation.

Example 2.17. Suppose $\mathcal{O}(B) = \{c,d,e,f,j\}$ and $\mathcal{O}(D) = \{d,e,f,g,h,i\}$, with three vertices $\{d,e,f\}$ in common. Since $\mathcal{O}(D) = \mathcal{O}(B) - \{c,j\} \cup \{g,h,i\}$, according to Proposition 2.6, the partial sums $Partial^{s_k}_{\mathcal{O}(D)}(\star)$ can be computed from the memoized $Partial^{s_k}_{\mathcal{O}(B)}(\star)$ as

$$\begin{aligned} Partial^{s_k}_{\mathcal{O}(D)}(\star) \;=\; & Partial^{s_k}_{\mathcal{O}(B)}(\star) + Partial^{s_k}_{\{g,h,i\}}(\star) \\ & - Partial^{s_k}_{\{c,j\}}(\star). \end{aligned} \tag{2.22}$$

However, in the context of partial max sharing, we may not obtain the partial max $Partial\_Max^{s_k}_{\mathcal{O}(D)}(\star)$ directly from the memoized $Partial\_Max^{s_k}_{\mathcal{O}(B)}(\star)$ via an Eq.(2.22)-like approach. This is because "subtraction" is involved in Eq.(2.22) — although we

[5] In the following, we shall focus solely on optimizing Eq.(2.17). A similar method can be applied to Eq.(2.18).

know

$$\begin{aligned} & Partial\_Max^{s_k}_{\mathcal{O}(B)\cup\{g,h,i\}}(\star) \\ =& \max\{Partial\_Max^{s_k}_{\mathcal{O}(B)}(\star), Partial\_Max^{s_k}_{\{g,h,i\}}(\star)\}, \end{aligned}$$

we do not know how to derive $Partial\_Max^{s_k}_{\mathcal{O}(D)}(\star)$ from $Partial\_Max^{s_k}_{\mathcal{O}(B)\cup\{g,h,i\}}(\star)$ and $Partial\_Max^{s_k}_{\{c,j\}}(\star)$, which is due to the "subtraction" curse in the context of max operator. □

This example tells that, for every two out-neighbor sets $\mathcal{O}(X)$ and $\mathcal{O}(Y)$, only when $\mathcal{O}(X) \subseteq \mathcal{O}(Y)$, then the partial max $Partial\_Max^{s_k}_{\mathcal{O}(X)}(\star)$ can be reused for computing $Partial\_Max^{s_k}_{\mathcal{O}(Y)}(\star)$ as

$$\begin{aligned} & Partial\_Max^{s_k}_{\mathcal{O}(Y)}(\star) \\ =& \max\{Partial\_Max^{s_k}_{\mathcal{O}(X)}(\star), Partial\_Max^{s_k}_{\mathcal{O}(Y)\setminus\mathcal{O}(X)}(\star)\}. \end{aligned}$$

Unfortunately, the condition $\mathcal{O}(X) \subseteq \mathcal{O}(Y)$ is too restrictive in real-life networks for partial max sharing. In practice, out-neighbors are often overlapped *irregularly* in many real-world graphs, *i.e.,* $\mathcal{O}(X) \cap \mathcal{O}(Y) \neq \varnothing$. It is imperative for us to find a new different way of partial max sharing, which can effectively avoid the "subtraction" curse for computing the Minimax SimRank variation.

Partial Max Sharing. The main idea of our approach is based on a *finer-grained* partial max sharing. Given two out-neighbor sets $\mathcal{O}(X)$ and $\mathcal{O}(Y)$, if $\mathcal{O}(X) \cap \mathcal{O}(Y) \neq \varnothing$, then we first memoize the finer-grained partial max over the common subset $\mathcal{O}(X) \cap \mathcal{O}(Y)$:

$$z(\star) = Partial\_Max^{s_k}_{\mathcal{O}(X)\cap\mathcal{O}(Y)}(\star), \tag{2.23}$$

then reuse $z(\star)$ to compute both $Partial\_Max^{s_k}_{\mathcal{O}(X)}(\star)$ and $Partial\_Max_{\mathcal{O}(Y)}(\star)$ as

$$\begin{aligned} Partial\_Max^{s_k}_{\mathcal{O}(X)}(\star) &= \max\{Partial\_Max^{s_k}_{\mathcal{O}(X)\setminus\mathcal{O}(Y)}(\star), z(\star)\}, \\ Partial\_Max^{s_k}_{\mathcal{O}(Y)}(\star) &= \max\{Partial\_Max^{s_k}_{\mathcal{O}(Y)\setminus\mathcal{O}(X)}(\star), z(\star)\}. \end{aligned}$$

In comparison, the partial sums sharing approach in Section 2.3, if ported to the partial max sharing, only allows $Partial\_Max^{s_k}_{\mathcal{O}(Y)}(\star)$ being computed from another memoized

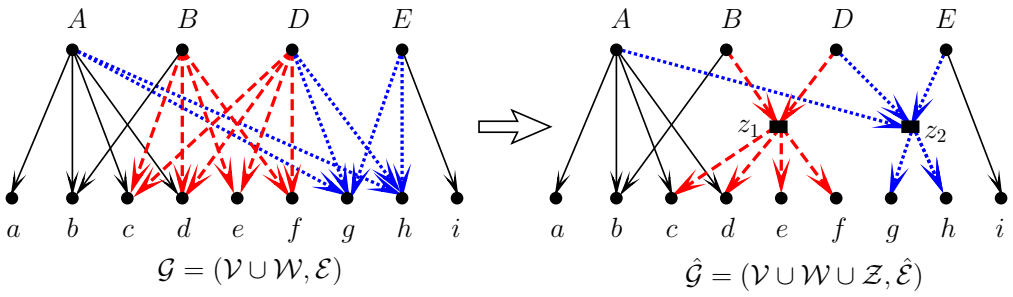


Figure 2.5: Edge Concentration

partial sums $Partial\_Max^{s_k}_{\mathcal{O}(X)}(\star)$ or from scratch (depending on the transition costs); since "subtraction" is not allowed in the context of max operator, $Partial\_Max^{s_k}_{\mathcal{O}(Y)}(\star)$ have to be calculated from scratch if $\mathcal{O}(X) \nsubseteq \mathcal{O}(Y)$. Fortunately, our approach can share the common subparts for both $Partial\_Max^{s_k}_{\mathcal{O}(X)}(\star)$ and $Partial\_Max^{s_k}_{\mathcal{O}(Y)}(\star)$ computation while preventing the "subtraction" curse.

Edge Concentration. To find out the common subparts $z(\star)$ in Eq.(2.23) for all out-neighbor sets sharing, we first introduce the notion of *biclique*.

Definition 2.18. Given a bipartite digraph $\mathcal{G} = (\mathcal{V} \cup \mathcal{W}, \mathcal{E})$, a pair of two disjoint subsets $(\mathcal{V}', \mathcal{W}')$, with $\mathcal{V}' \subseteq \mathcal{V}$ and $\mathcal{W}' \subseteq \mathcal{W}$, is called a biclique if $(v', w') \in \mathcal{E}$ for all $v' \in \mathcal{V}'$ and $w' \in \mathcal{W}'$. □

Clearly, a biclique $(\mathcal{V}', \mathcal{W}')$ is a complete subgraph in the bipartite digraph $\mathcal{G} = (\mathcal{V} \cup \mathcal{W}, \mathcal{E})$, denoting the densest parts in $\mathcal{G}$. For example in the left part of Figure 2.5, $(\{B, D\}, \{c, d, e, f\})$ (dashed arrows) and $(\{A, D, E\}, \{g, h\})$ (dotted arrows) are two bicliques.

Bicliques are utilized for finding out the common subparts for partial max sharing. A biclique, say $(\{B, D\}, \{c, d, e, f\})$, in $\mathcal{G}$ means that the out-neighbor sets $\mathcal{O}(B)$ and $\mathcal{O}(D)$ have common vertices $\{c, d, e, f\}$. Thus, $Partial\_Max^{s_k}_{\{c,d,e,f\}}(\star)$ can be reused for both $Partial\_Max^{s_k}_{\mathcal{O}(B)}(\star)$ and $Partial\_Max^{s_k}_{\mathcal{O}(D)}(\star)$ computation. Pictorially, such a partial max sharing optimization process can be depicted by the *edge concentration* [Lin00] of a biclique in $\mathcal{G}$. As shown in the right part of Figure 2.5, after edge concentration, a

biclique, say $(\{B, D\}, \{c, d, e, f\})$, can be simplified into a triple $(\{B, D\}, z_1, \{c, d, e, f\})$, where we call $z_1 \in \mathcal{Z}$ a *concentration vertex*. Each triple, say $(\{B, D\}, z_1, \{c, d, e, f\})$, tells us the following: (1) First, all the out-neighbors of vertex z_1 can be clustered together to produce the memoized results $z_1(\star)$, *i.e.,*

$$z_1(\star) = Partial\_Max^{s_k}_{\{c,d,e,f\}}(\star).$$

(2) Then, each in-neighbor of vertex z_1, say B, indicates that the memoized $z_1(\star)$ can be reused in partial max computation $Partial\_Max^{s_k}_{\mathcal{O}(B)}(\star)$, *i.e.,*

$$Partial\_Max^{s_k}_{\mathcal{O}(B)}(\star) = \max\{Partial\_Max^{s_k}_{\{b\}}(\star), z_1(\star)\}.$$

Therefore, applying edge concentration to every biclique of $\mathcal{G}$ provides a very efficient way for partial max sharing. The main advantage is that, after edge concentration, the number of edges in every biclique $(\mathcal{V}', \mathcal{W}')$ can be reduced from $|\mathcal{V}'| \times |\mathcal{W}'|$ to $|\mathcal{V}'| + |\mathcal{W}'|$. It is worth mentioning that for every fixed vertex x, the total cost of performing the partial max $Partial\_Max^{s_k}_{\mathcal{O}(\star)}(x)$ over all out-neighbor sets $\mathcal{O}(\star)$ is equal to the number $|\mathcal{E}|$ of edges in $\mathcal{G}$. Hence, our goal of minimizing the total cost of the partial max is equivalent to the problem of minimizing the number of edges in $\mathcal{G}$ via edge concentration. However, this problem is NP-hard, as proved in our early work [LHH+10]. Thus, to find bicliques in $\mathcal{G}$, we invoke a heuristic [BC08b].

Algorithm. We next present an algorithm for computing Minimax SimRank variation in a bipartite graph.

The algorithm, max-MSR, is shown in Algorithm 2.2. It takes as input the bipartite graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{W}, \mathcal{E})$, a damping factor C, and the number of iterations K, and returns all pairs of Minimax SimRank similarities.

The algorithm max-MSR runs in three phases.

(1) Precomputing (lines 1–5). The algorithm first finds bicliques in bipartite graph $\mathcal{G}$ by invoking the algorithm in [BC08b] (line 1). It then replaces all the bicliques (densest parts) in $\mathcal{G}$ via edge concentration (lines 2–5).

Algorithm 2.2: max-MSR $(\mathcal{G}, C, K)$

Input : bipartite graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{W}, \mathcal{E})$, damping factor C, the number of iterations K.

Output: all the similarities of Minimax SimRank variation $s_K(\star, \star)$.

find all the bicliques $\{(\mathcal{V}'_i, \mathcal{W}'_i)\}$ in $\mathcal{G}$;
foreach *biclique* $(\mathcal{V}'_i, \mathcal{W}'_i)$ *in* $\mathcal{G}$ **do**
    Delete all the edges $(v', w') \in \mathcal{V}'_i \times \mathcal{W}'_i$;
    Insert a dummy vertex z_i into $\mathcal{Z}$;
    Add edges $(v', z_i), (z_i, w'),\ \forall v' \in \mathcal{V}', w' \in \mathcal{W}'$;
initialize $s_0(A, B) \leftarrow \begin{cases} 1, & A=B \\ 0, & A \neq B \end{cases} \quad \forall A, B \in \mathcal{V}$;
for $k \leftarrow 0, 1, \cdots, K-1$ **do**
    foreach *vertex* $i \in \mathcal{V}$ *in* $\mathcal{G}$ **do**
        foreach *dummy vertex* $z_j \in \mathcal{Z}$ **do**
            $s_k(i, z_j) \leftarrow \max_{x \in \mathcal{O}(z_j)} s_k(i, x)$;
        foreach *vertex* $B \in \mathcal{V}$ *in* $\mathcal{G}$ **do**
            $Partial\_Max^{s_k}_{\mathcal{O}(B)}(i) \leftarrow \max_{x \in \mathcal{O}(B)} s_k(i, x)$;
    foreach *dummy vertex* $z_j \in \mathcal{Z}$ **do**
        foreach *vertex* $B \in \mathcal{V}$ *in* $\mathcal{G}$ **do**
            $Partial\_Max^{s_k}_{\mathcal{O}(B)}(z_j) \leftarrow \sum_{x \in \mathcal{O}(z_j)} Partial\_Max^{s_k}_{\mathcal{O}(B)}(x)$;
    foreach *vertex* $B \in \mathcal{V}$ *in* $\mathcal{G}$ **do**
        foreach *vertex* $A \in \mathcal{V}$ *in* $\mathcal{G}$ **do**
            if $A=B$ **then** $s_{k+1}(A, B) = 1$; continue;
            if $\mathcal{O}(A) = \varnothing$ **then** $s^A_{k+1}(A, B) \leftarrow 0$;
            else $s^A_{k+1}(A, B) \leftarrow \frac{C}{|\mathcal{O}(A)|} \sum_{i \in \mathcal{O}(A)} Partial\_Max^{s_k}_{\mathcal{O}(B)}(i)$;
            if $\mathcal{O}(B) = \varnothing$ **then** $s^B_{k+1}(A, B) \leftarrow 0$;
            else $s^B_{k+1}(A, B) \leftarrow \frac{C}{|\mathcal{O}(B)|} \sum_{i \in \mathcal{O}(B)} Partial\_Max^{s_k}_{\mathcal{O}(A)}(i)$;
            $s_{k+1}(A, B) \leftarrow \min\{s^A_{k+1}(A, B), s^B_{k+1}(A, B)\}$;
    free $Partial\_Max^{s_k}_{\mathcal{O}(\star)}(\star)$;
return $s_K(\star, \star)$;

(2) Inner Partial Max Sharing (lines 8–12). The algorithm then iteratively computes the common subparts among the different $Partial\_Max^{s_k}_{\mathcal{O}(\star)}(\star)$ (lines 9–10). Once computed, the finer-gained inner partial max results are memoized for computing all the partial max over different out-neighbor sets (lines 11–12).

(3) Outer Partial Sums Sharing (lines 13–22). The algorithm next computes the common subparts among the different outer partial sums (lines 13–15). Once computed, the finer-gained outer partial sums results are memoized for computing all the similarities of Minimax SimRank $s_{k+1}(\star,\star)$ (lines 16–21). After every iteration, the partial max results can be removed from memory (line 22).

Correctness & Complexity. One can readily verify that the algorithm correctly computes $s_K(\star,\star)$, which satisfies Eqs.(2.17)–(2.19).

The time of max-MSR is bounded by $O(Km'n)$, where

$$m' = |\mathcal{E}| - \sum_{i=1}^{N} (|\mathcal{V}'_i| \times |\mathcal{W}'_i| - |\mathcal{V}'_i| - |\mathcal{W}'_i|),$$

with N being the total number of bicliques $(\mathcal{V}'_i, \mathcal{W}'_i)$ in the bipartite graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{W}, \mathcal{E})$. Here, $m' \leq |\mathcal{E}|$, and in practice, m' is much smaller than $|\mathcal{E}|$ since there could be many small dense parts in real bipartite graphs.

We analyze the time complexity in detail below. The total time of max-MSR consists of three phases: precomputing, inner partial max sharing, and outer partial sums sharing.

(1) For the precomputing (lines 1–5), a heuristic algorithm in [BC08b] is leveraged for finding bicliques in $\mathcal{G}$, which requires $O(|\mathcal{E}| \log(|\mathcal{V}| + |\mathcal{W}|))$ time.

(2) In the inner partial max sharing phase (lines 8–12), for every iteration k and each fixed vertex i, the total cost of computing $Partial\_Max^{s_k}_{\mathcal{O}(\star)}(i)$ is equal to the number of edges in the reduced graph of $\mathcal{G}$ via edge concentration, which is $O(m')$. This is because replacing each biclique can reduce the cost of max operations from $|\mathcal{V}'_i| \times |\mathcal{W}'_i|$ to $|\mathcal{V}'_i| + |\mathcal{W}'_i|$. Thus, for N bicliques in $\mathcal{G}$, $O(\sum_{i=1}^{N} (|\mathcal{V}'_i| \times |\mathcal{W}'_i| - |\mathcal{V}'_i| - |\mathcal{W}'_i|))$ time is reduced. Hence, for K iterations, computing all the partial max over all the out-neighbor sets requires $O(Km'n)$ time.

Dataset		Vertices	Edges	Avg Deg.
BERKSTAN		685,230	7,600,595	11.1 (in)
PATENT		3,774,768	16,518,948	4.4 (in)
COURSE		8,470+1,873	46,825	5.53 (out)
IMDB		320.1K+785.6K	3,871,636	12.09 (out)
DBLP	D02	9,942	27,849	2.8 (in)
	D05	15,976	38,356	2.4 (in)
	D08	23,471	63,723	2.7 (in)
	D11	39,965	104,468	2.6 (in)

Figure 2.6: Real-life Dataset Details

(3) For the outer partial sums sharing (lines 13–22), similar to the partial max sharing phase, the cost of computing all similarities $s_K(\star,\star)$ from the memoized $Partial\_Max^{s_k}_{\mathcal{O}(\star)}(\star)$ is equal to the number of of edges in the reduced graph of $\mathcal{G}$, entailing $O(Km')$ time for K iterations.

Taking the three phases together, the total cost of max-MSR is dominated by the second phase, which is in $O(Km'n)$ time.

2.6 Empirical Evaluation

We present an experimental study on real and synthetic data to evaluate the efficacy of our methods.

2.6.1 Experimental Setting

Datasets. For the basic SimRank model, we use three real datasets (BERKSTAN, PATENT, DBLP) to evaluate the efficiency of our approaches, and one synthetic dataset (SYN) to vary graph characteristics. For the Minimax SimRank variation model in bipartite domains, we use two real datasets (COURSE and IMDB) and one syntectic bipartite dataset (SYNBI).

The sizes of the datasets are illustrated in Figure 2.6. In the following, we provide a detailed description of these datasets.

(1) BerkStan. The first network is a Berkeley-Stanford web graph of 7.4M links between 680K web pages (from `berkely.edu` and `stanford.edu` domains), downloaded from the Stanford Network Analysis Project (SNAP).[6]

(2) Patent. This is a citation network among U.S. Patents, obtained from the National Bureau of Economic Research. [7] It is our largest dataset consisting of 3.2M U.S. patents (vertices) and 16.1M citations (edges), with a low average degree of 4.4.

(3) DBLP. This is a scientific publication network, derived from DBLP Computer Science Bibliography.[8] We selected the recent 12-year publications (from 2000 to 2011) in 8 major conferences (ICDE, VLDB, SIGMOD, PODS, CIKM, ICDM, SIGIR, SIGKDD), and then built 4 co-authorship graphs by choosing every 3 years as a time step.

(4) Course. This dataset is obtained from the transcripts of 8,470 students in the University of New South Wales. Every transcript lists the courses that the student has taken. There are 1,873 courses in total, with an average of about 25 courses for each student.

(5) IMDB. The IMDB network[9] is a bipartite graph, with two types of vertices: 20.1K movies and 785.6K actors. Each edge from a movie to an actor means that the actor name movies has appeared in the movie. There are 3.8M edges in this dataset, among with 8,695 edges are multiple edges. For our Minimax SimRank analysis, we treated multiple edges as single ones.

(6) Syn. The synthetic data were produced by the graph generator GTGraph [10], varying two parameters: the number of vertices, and the number of edges. We generated the graphs following the power laws.

(7) SynBI. The synthetic bipartite graphs were also generated by GTGraph, denoted as SynBI, with vertex sets of two sides having one half of the vertices, and edges being randomly generated.

[6] http://snap.stanford.edu/data/web-BerkStan.html
[7] http://data.nber.org/patents/
[8] http://dblp.uni-trier.de/~ley/db/
[9] http://www.imdb.com
[10] http://www.cse.psu.edu/~madduri/software/GTgraph/

Compared Algorithms. We implement 7 algorithms via Visual C++ 8.0. (1) OIP-DSR, our differential SimRank of Eq.(2.15)[11] in conjunction with partial sums sharing. (2) OIP-SR, our basic SimRank using partial sums sharing. (3) psum-SR [LVGT10], without partial sums sharing. (4) mtx-SR [LHH+10], a matrix-based SimRank via SVD factorization. (5) max-MSR, our bipartite Minimax SimRank variation using finer-grained partial max sharing. (6) psum-MSR, the baseline bipartite Minimax SimRank variation, with partial max sharing via Eq.(2.20). (7) MSR [JW02, Section 4.3.1], the original iterative bipartite Minimax SimRank variation.

We set the following default parameters as used in [LVGT10]: $C = 0.6, \epsilon = 0.001$ (unless otherwise mentioned). For all the methods, we clip similarity values at 0.001, to discard far-apart nodes with scores less than 0.001 for storage. It can significantly reduce space cost with minimal impact on accuracy, as shown in [LVGT10].

Evaluation Metrics. To evaluate ranking results on DBLP, we used *Normalized Discounted Cumulative Gain* (NDCG) [LHH+10]. The NDCG at rank position p is defined as follows

$$\text{NDCG}_p = \frac{1}{\text{IDCG}_p} \sum_{i=1}^{p} (2^{\text{rank}_i} - 1)/\log_2(1+i),$$

where rank_i is the graded relevance at position i, and IDCG_p is a normalization factor, ensuring the NDCG of an ideal ranking at position p is 1.

For ground truth, we invited twelve independent evaluators from the database community, and used their final judgment, rendered by a majority vote, as the standard. To validate the relative order of co-authors for different algorithms on DBLP, these experts may assess the "true" relevance of each retrieved co-authorship, by referring to Co-Author Path in Microsoft Academic Search[12] to see "separations" between collaborators.

We used a machine powered by a Quad-Core Intel i5 CPU (3.10GHz) with 16GB RAM, using Windows 7. Each experiment was run 5 times, and the average performance is reported here.

[11] In the previous conference version [YLZ+13b], OIP-DSR is our differential SimRank of Eq.(2.14), which requires more memory space for storing the intermediate results.

[12] http://academic.research.microsoft.com/VisualExplorer

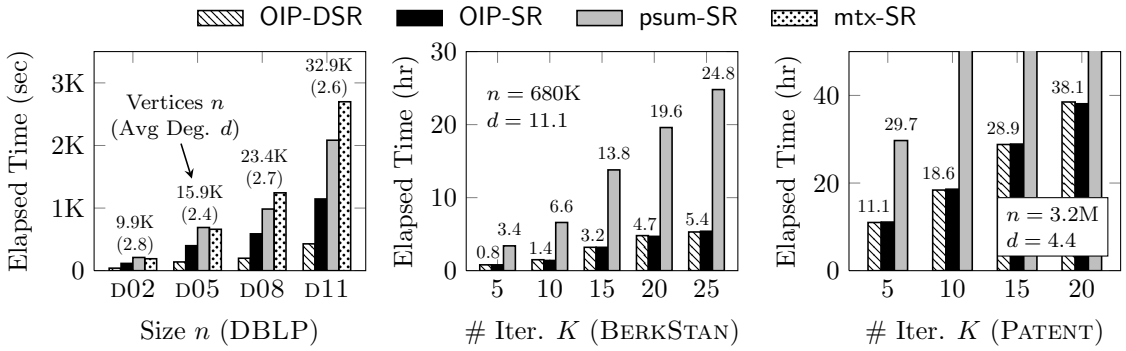


Figure 2.7: Time Efficiency on Real Datasets

2.6.2 Experimental Results

Exp-1: Time Efficiency.

We first evaluate (1) the CPU time of OIP-SR and OIP-DSR on real data, and (2) the impact of graph density on CPU time, using synthetic data. To favor mtx-SR that only works on *low-rank graphs* (*i.e.,* graph with a small rank of the adjacency matrix), DBLP data are used although OIP-SR and OIP-DSR work pretty well on various graphs.

Fixing the accuracy $\epsilon = .001$ for DBLP, varying K for BerkStan and Patent, Figure 2.7 compare the CPU time of the four algorithms. (1) In all the cases, OIP-SR consistently outperforms mtx-SR and psum-SR, *i.e.,* our partial sums sharing approach is effective. On BerkStan and Patent, the speedups of OIP-SR are on average 4.6X and 2.7X, respectively, better than psum-SR. On the large Patent, when $K \geq 8$, psum-SR takes too long to finish the computation in two days, which is practically unacceptable. In contrast, OIP-SR and OIP-DSR just need about 18.6 hours for $K = 10$. (2) OIP-DSR always runs up to 5.2X faster than psum-SR, and 3X faster than OIP-SR on DBLP, for the desired $\epsilon = .001$. This is because the differential matrix form of OIP-DSR increases the rate of convergence, which enables fewer iterations for attaining the given ϵ. (3) The speedups of OIP-SR and OIP-DSR on BerkStan (4.6X) are more pronounced than those on DBLP (1.8X) and Patent (2.7X), which is due to the high degree of BerkStan ($d = 11.1$) that may potentially increase the overlapped area for common in-neighbor sets, and thus provides more opportunities for partial sums sharing.

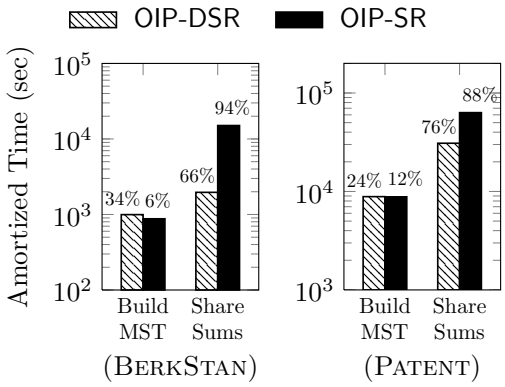


Figure 2.8: Amortized Time on Real Data

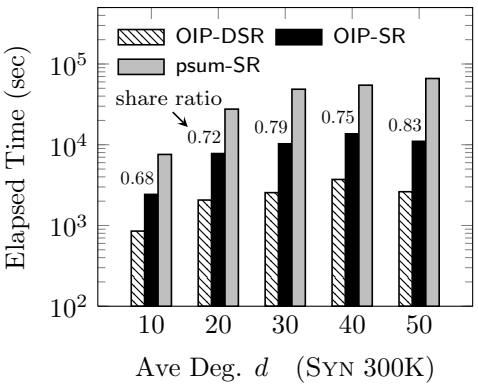


Figure 2.9: Effect of Density

Figure 2.8 further shows the amortized time for each phase of OIP-SR and OIP-DSR on BerkStan and Patent data (given $\epsilon = .001$), in which x-axis represents different stages. From the results, we can discern that (1) for OIP-SR, the time taken for "Building MST" is far less than the time taken for "Share Sums". This confirms our complexity analysis in Proposition 2.10. (2) "Building MST" always takes up larger portions (34% on BerkStan, and 24% on Patent) in the total time of OIP-DSR, than those (6% on BerkStan, and 12% on Patent) in the total time of OIP-SR. This becomes more evident on various datasets because OIP-SR and OIP-DSR takes (almost) the same time for "Building MST", whereas, for "Sharing Sums", OIP-DSR enables less time (4.5X on BerkStan, and 2.5X on Patent) than OIP-SR, due to the speedup in the convergence rate of OIP-DSR.

Fixing $n = 300$K and varying m from 3M to 15M on the synthetic data, Figure 2.9 reports the impact of graph density (ave. in-degree) on CPU time, where y-axis is in the log scale. The results show that (1) for $\epsilon = .001$, OIP-DSR significantly outperforms psum-SR by at least one order of magnitude as m increases. In all the cases, OIP-SR achieves 0.5 order of magnitude speedups on average. (2) Interestingly, the speedups of OIP-DSR are sensitive to graph density (ave. in-degree d) The larger the d is, the higher the likelihood of overlapping in-neighbors is for partial sums sharing, as expected. The biggest speedups are observed for larger d (higher density) — with nearly 2 orders of magnitude speedup for $d = 50$.

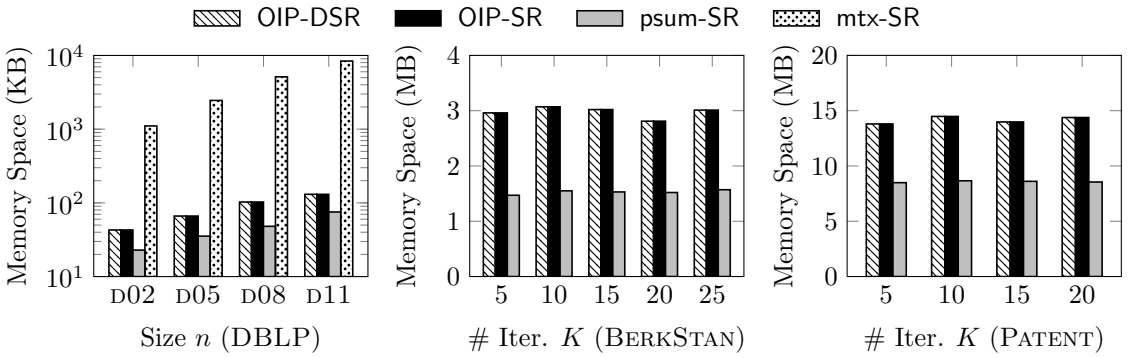
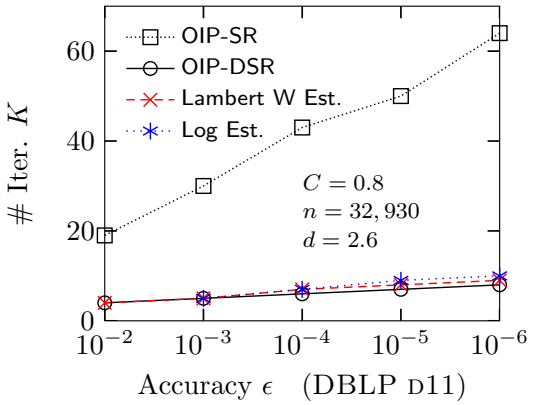


Figure 2.10: Memory Space on Real Datasets

Exp-2: Memory Space.

We next evaluate the memory space efficiency of OIP-DSR and OIP-SR on real data. Note that we only use mtx-SR on small DBLP as a baseline; for large BerkStan and Patent, the memory space of mtx-SR will explode as the SVD method of mtx-SR destroys the graph sparsity.

Figure 2.10 shows the results on space. We observe that (1) on DBLP, OIP-DSR and OIP-SR have much less space than mtx-SR by at least one order of magnitude, as expected. (2) In all the cases, the space cost of OIP-DSR and OIP-SR fairly retains the same order of magnitude as psum-SR. Indeed, both OIP-DSR and OIP-DSR merely need about 1.8X, 1.9X, 1.6X space of psum-SR on DBLP, BerkStan, Patent, respectively, for outer partial sums sharing. This confirms our complexity analysis in Section 2.3, suggesting that OIP-DSR and OIP-DSR do not require too much extra space for caching outer partial sums. Moreover, OIP-DSR has almost the same space as OIP-SR since Eq.(2.15) does not need to memoize the auxiliary $\mathbf{T}_k$ in Eq.(2.14). (3) On BerkStan and Patent, the space costs of OIP-DSR and OIP-SR are stabilized as K increases. This is because the memoized partial sums are released immediately after each iteration, thus maintaining the same space during the iterations.



(a) Accuracy *vs.* #-Iter.

Err ϵ	OIP-SR	OIP-DSR	LamW Est.	Log Est.
10^{-2}	19	4	4	-
10^{-3}	30	5	5	5
10^{-4}	43	6	7	7
10^{-5}	50	7	8	9
10^{-6}	64	8	9	10

(b) Lam W & Log Bound on K

Figure 2.11: Convergence Rate

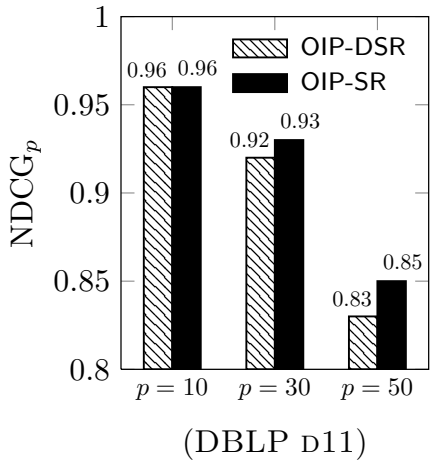
Exp-2: Memory Space.

We next compare the convergence rate of OIP-DSR and OIP-SR, using real and synthetic data. For the interest of space, below we only report the results on DBLP D11 ($C = 0.8$). The trends on other datasets are similar.

By varying ϵ from 10^{-2} to 10^{-6}, Figs. 2.11a and 2.12 show that (1) OIP-DSR needs far fewer iterations than OIP-SR (also psum-SR), for a given accuracy. Even for a small $\epsilon = 10^{-6}$, OIP-DSR only requires 8 iterations, whereas the convergence of OIP-SR in this case becomes sluggish, yielding over 60 iterations. This confirms our observation in Proposition 2.14 that OIP-DSR has an exponential rate of convergence. (2) The two curves labeled "Lambert W Est." and "Log Est." (dashed line) visualize our apriori estimates of K' derived from Corollaries 2.15 and 2.16, respectively. We can see that these dashed curves are close to the actual number iterations of OIP-DSR, suggesting that our estimates of K' for OIP-DSR are fairly precise.

Exp-4: Relative Order.

To analyze the relative order of the similarities from OIP-DSR and OIP-SR, we use DBLP D11, a co-authorship graph with ground truth. Fixing a vertex a as a given query (author), we compute the NDCG$_p$ of OIP-DSR and OIP-SR via the similarities $s(a, \star)$



(DBLP D11)

Figure 2.12: Relative Ordering

#	Co-authors	#	Co-authors	#	Co-authors
1	Hongjun Lu	11	James Cheng	21	Wenfei Fan
2	Lu Qin	12	Weifa Liang	22	Rong-Hua Li
3	Xuemin Lin	13	Ying Zhang	23	**Hong Cheng ▼**
4	Wei Wang	14	Bolin Ding	24	**Jun Gao ▲**
5	Lei Chen	15	Haixun Wang	25	Xiaofang Zhou
6	Lijun Chang	16	Aoying Zhou	26	Ke Yi
7	Yiping Ke	17	Xiang Lian	27	Yufei Tao
8	Haifeng Jiang	18	Cheqing Jin	28	Nan Tang
9	Philip S. Yu	19	Baichen Chen	29	Jinsoo Lee
10	Gabriel Pui Cheong Fung	20	Byron Choi	30	Kam-Fai Wong

Figure 2.13: Case Study: Co-authors of "Jeffrey Xu Yu"

from the top-p query perspective. For query selection, we sort all the vertices in order of their degree into 4 groups, and then randomly choose 100 vertices from each group, in order to ensure that the selected vertices can systematically cover a broad range of all possible queries. For each query, Figure 2.12 compares the average NDCG$_p$ values of OIP-DSR with its counterparts of OIP-SR, for $p = 10, 30, 50$. The result shows that OIP-DSR can perfectly maintain the relative order of the similarity scores produced by OIP-DSR with only 1% loss of NDCG$_{30}$ and NDCG$_{50}$. For $p = 10$ (*i.e.,* top-10 query), OIP-DSR produces exactly the same result of OIP-SR, as expected. Thus, we can gain a lot in speedup from OIP-DSR while suffering little loss in quality.

For case study, Figure 2.13 shows the top-30 co-authors of "Prof. Jeffrey Xu Yu" via OIP-DSR on DBLP D11. The results of OIP-DSR, as compared as with OIP-SR,

only differ in one inversion at two adjacent positions (#23, #24), which is practically acceptable. This confirms our intuitions in Section 2.4, where we envisage that slightly modifying the damping factor in OIP-DSR never incurs high quality loss.

Exp-5: Minimax SimRank Variation.

Finally, we evaluate the computational time and memory space of max-MSR against the baseline psum-MSR and MSR on bipartite real networks (Course and IMDB) and synthetic dataset (SynBI).

To compare the CPU time of the three Minimax SimRank variations, on Course and IMDB, we vary K from 5 to 25; on SynBI, we fix $n = 200$K with each side of the bipartite graph having 100K vertices, and vary the average out-degree from 5 to 35. The results are reported in Figure 2.14. (1) In all the cases, max-MSR is always the fastest, and psum-MSR the second, both of which significantly outperform MSR by several times on Course and by one order of magnitude on IMDB. This is because partial max memoization can achieve high speedups for Minimax SimRank computation. Moreover, the finer-grained partial max memoization of max-MSR can share much more common subparts that are neglected by psum-MSR. Thus, max-MSR is consistently better than psum-MSR. On large IMDB, the speedup is more apparent, *e.g.,* for $K = 5$, the time of max-MSR (0.6hr) is 5.15X faster than psum-MSR (3.2hr); however, it takes too long time for MSR to finish the computation within one day. Hence, we stop iterating after $K \geq 5$ iterations on psum-MSR and $K \geq 15$ on SynBI, respectively. (2) The graph density has a huge impact on the speedup of max-MSR. The denser the graph, the more likely the common out-neighbors (bicliques) can be shared for partial max memoization. This explains why the reduced amount of time for max-MSR relative to psum-MSR is far more pronounced on IMDB than on Course, as IMDB has a higher average out-degree (12.09) than Course (5.53). The results on SynBI have also confirmed this observation, where we notice that the share ratio tends to increase *w.r.t.* the growing average out-degree of the synthetic graph.

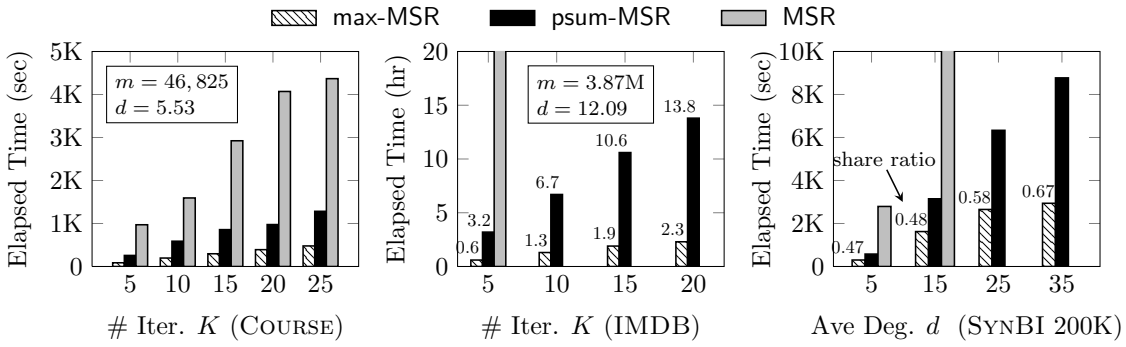


Figure 2.14: Time Efficiency on Bipartite Networks

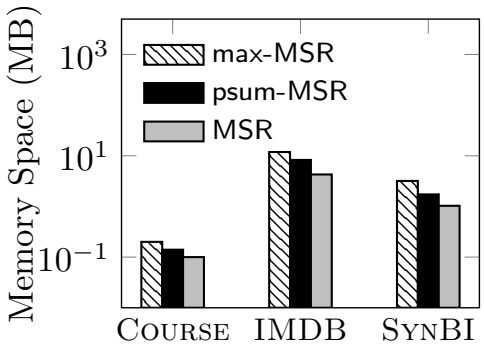


Figure 2.15: Memory Space on Bipartite Networks

The memory space of these Minimax SimRank variations on real and synthetic datasets is evaluated in Figure 2.15. Due to space limitations, we merely report the results on SYNBI with the average out-degree of 25. We notice that in all the cases, the memory space of max-MSR is a bit higher than that of psum-MSR, both of which are a bit higher than MSR, yet maintain the same order of magnitude during the iterations. For instance on IMDB, the space cost for max-MSR (0.2M) is slightly higher than psum-MSR (0.14M) and MSR (0.10M). This is because the partial max memoization requires extra space to cache similarities of all dummy vertices. The finer the granularity for memoization, the more space it requires, as expected.

2.7 Related Work

The development of efficient methods to compute SimRank is a vibrant research area [YZL+12, LVGT10, LHH+10] that is fundamental to *e.g.,* web mining and object ranking.

Recent results on SimRank can be summarized as follows.

The earliest mention of SimRank dates back to Jeh and Widom [JW02] who suggested (i) an iterative approach to compute SimRank, which is in $O(Kd^2n^2)$ time, along with (ii) a heuristic pruning rule to set the similarity between far-apart vertices to be zero. Unfortunately, the naive iterative SimRank is rather costly to compute, and there is no provable guarantee on the accuracy of the pruning results. To overcome the limitations, a very appealing attempt was made by Lizorkin *et al.* [LVGT10] who (i) provided accuracy guarantees for SimRank iterations, *i.e.,* the number of iterations needed for a given accuracy ϵ is $K = \lceil \log_C \epsilon \rceil$, and (ii) proposed three excellent optimization approaches, *i.e.,* essential node-pair selection, partial sums memoization, and threshold-sieved similarities. Especially, partial sums memoizing serves as the cornerstone of their strategies, which significantly reduces the computation of SimRank to $O(Kdn^2)$ time. Our work differs from [LVGT10] in the following. (i) We put forward the phenomenon of partial sums redundancy in [LVGT10] that typically exists in real graphs. (ii) We accelerate the convergence of SimRank iterations from geometric [LVGT10] to exponential growth, by revising the existing SimRank model. (iii) In bipartite domains, we also devise a partial max sharing for the Minimax SimRank variation model.

There has also been a flurry of research interests (*e.g.,* [LHH+10, HFLC10, ZHS09, AMC08, FR07, LHH+10, LLY12]) in the SimRank optimization problems. Li *et al.* [LHH+10] first based SimRank computation on the matrix representation. They developed very interesting SimRank approximation techniques on a low-rank graph, by leveraging the singular value decomposition and tensor product. However, (i) for digraphs, the upper bound of approximation error still remains unknown. (ii) The computational time in [LHH+10] would become $O(n^4)$ even when the rank of an adjacency matrix is relatively small, *e.g.,* $\lceil \sqrt{n} \rceil$ ($\ll n$). The pioneering work of He *et al.* [HFLC10] utilized the node-updating method on GPU for parallel SimRank computing. They deployed iterative aggregation techniques to accelerate the global convergence of parallel SimRank, in which the speed-up in the global convergence of SimRank is due mainly to the different

local convergence rates on small matrix partitions. Recently, the new notions of weight- and evidence-based SimRank have been suggested in [AMC08] to address the issue of query rewriting for sponsored search. Fogaras *et al.* [FR07] adopted a scalable Monte Carlo sampling approach to estimate SimRank by using the first meeting time of two random surfers. However, their algorithms are probabilistic in nature. Li *et al.* [LHH+10] employed an effective method for locally computing single-pair SimRank by breaking the holistic nature of the SimRank recursion. Zhao *et al.* [ZHS09] proposed a new ranking model, termed Penetrating Rank (P-Rank), by taking account of both in- and out-links. Since the iterative paradigms of SimRank and P-Rank are almost similar, our techniques for SimRank can be easily extended to P-Rank computation. Lee *et al.* [LLY12] devised a top-K SimRank algorithm needing to access only a small fraction of vertices in a graph. Most recently, Fujiwara *et al.* [FNSO13] proposed an excellent SVD-based SimRank for efficiently finding the top-k similar nodes *w.r.t.* a query.

2.8 Conclusions

In this chapter, we have proposed three efficient methods to speed up the computation of SimRank on large networks and bipartite domains. Firstly, we leveraged a novel clustering approach to optimize partial sums sharing. By eliminating the duplicates of computational efforts among the partial summations, an efficient algorithm was devised to greatly reduce the time complexity of SimRank. Secondly, we proposed a revised SimRank model based on the matrix differential representation, achieving an exponential speedup in the convergence rate of SimRank, as opposed to its conventional counterpart of a geometric speedup. Thirdly, in bipartite domains, we developed a novel finer-grained partial max clustering method for greatly accelerating the computation of the Minimax SimRank variation, and showed that the partial max sharing approach is different from the partial sums sharing method in that the "subtraction" is disallowed in the context of max operation. Our empirical experiments on both real and synthetic datasets have shown that the integration of our proposed methods for the basic SimRank equation

can significantly outperform the best known algorithm by about one order of magnitude, and that the computational time of our finer-grained partial max sharing method for the Minimax SimRank variation in bipartite domains outperforms the baselines by 0.5–1.2 orders of magnitude.

Chapter 3

Incremental SimRank on Link-Evolving Graphs

3.1 Introduction

With many recent eye-catching advances of the Internet, link analysis has become a common and important tool for web data management. Due to the proliferative applications (*e.g.,* link prediction, recommender systems, citation analysis), SimRank has stood out as an arresting one over the last decade, due to its succinct and iterative philosophy that "two nodes are similar if they are referenced by similar nodes", coupled with the base case that "every node is maximally similar to itself". In Chapter 2, the batch computation of SimRank on static networks has been investigated, which requires $O(Kd'n^2)$ time for all node-pairs, where K is the number of iterations, and $d' \leq d$ (d is the average graph in-degree).

In general, real graphs are often large, with links constantly evolving with minor changes. This is particularly evident in *e.g.,* co-citation networks, web graphs, and social networks. As a statistical example [NCO04], there are 5%–10% links updated every week in a web graph. It is rather expensive to reassess similarities for all pairs of nodes from scratch when the graph is updated. Fortunately, we observe that when link updates are

small, the affected areas for SimRank updates are often small as well. With this comes the need for incremental algorithms computing changes to SimRank in response to link updates, to skip unnecessary recomputations.

3.1.1 Problem Statement

Motivated by this, in this chapter we investigate the following problem for SimRank assessment.

Problem (Incremental SimRank Computation)

Given a network G, the similarities $\mathbf{S}$ for G, the link changes ΔG [1] to G, and the damping factor $C \in (0, 1)$.

Compute the changes $\mathbf{\Delta S}$ to the similarities $\mathbf{S}$.

In contrast with the work on batch SimRank computation, the study on incremental SimRank for link updates is limited. Indeed, due to the recursive nature of SimRank, it is hard to identify "affected areas" for incrementally updating SimRank. To the best of our knowledge, there is only one work [LHH+10] by Li *et al.* who gave a pioneering method for finding the SimRank changes in response to link updates. Their idea is to factorize the backward transition matrix $\mathbf{Q}$ [2] of the original graph into $\mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$ [3] via the singular value decomposition (SVD) first, and then incrementally estimate the updated matrices of $\mathbf{U}$, $\mathbf{\Sigma}$, $\mathbf{V}^T$ for link changes at the expense of exactness. As a result, updating the similarities of all node-pairs entails $O(r^4 n^2)$ time without guaranteed accuracy, where r ($\leq n$) is the target rank of the low-rank approximation[4], which is not always negligibly small in practice, as illustrated in the following example.

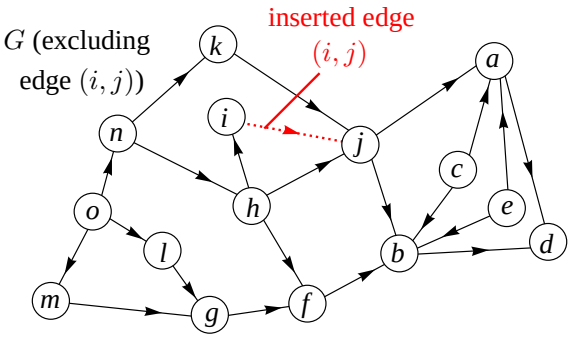
Example 3.1. Figure 3.1 is a citation graph G (a fraction of DBLP) where each edge depicts a reference from one paper to another. Assume G is updated by adding an edge

[1] ΔG consists of a sequence of edges to be inserted/deleted.

[2] In the notation of [LHH+10], the backward transition matrix $\mathbf{Q}$ is denoted as $\tilde{\mathbf{W}}$, which is the row-normalized transpose of the adjacency matrix.

[3] We use $\mathbf{X}^T$ (instead of $\tilde{\mathbf{X}}$ in [LHH+10]) to denote the transpose of matrix $\mathbf{X}$.

[4] According to [LHH+10], using our notations, $r \leq \text{rank}(\mathbf{\Sigma} + \mathbf{U}^T \cdot \mathbf{\Delta Q} \cdot \mathbf{V})$, where $\mathbf{\Delta Q}$ is the changes to $\mathbf{Q}$ for link updates.



Node-Pair	in G	in $G \cup \Delta G$	
	sim	sim_{true}	$\text{sim}_{\text{Li et al.}}$
(a, b)	0.075	0.062	0.073
(a, d)	0.000	0.006	0.002
$\boldsymbol{(i, f)}$	**0.246**	**0.246**	**0.246**
$\boldsymbol{(k, g)}$	**0.128**	**0.128**	**0.128**
$\boldsymbol{(k, h)}$	**0.288**	**0.288**	**0.288**
(j, f)	0.206	0.138	0.206
$\boldsymbol{(m, l)}$	**0.160**	**0.160**	**0.160**
(j, b)	0.000	0.030	0.001

Figure 3.1: Compute SimRank incrementally as edge (i, j) is added

(i, j), denoted by ΔG (see the dash arrow). Using the damping factor $C = 0.8$ [5], we want to compute SimRank scores in the new graph $G \cup \Delta G$. The existing method by Li *et al.* (see Algorithm 3 in [LHH+10]) first decomposes the old $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$ as a precomputation step, then, when edge (i, j) is added, it incrementally updates the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^T$, and utilizes their updated versions to obtain the new SimRank scores in $G \cup \Delta G$. The results are shown in Column '$\text{sim}_{\text{Li et al.}}$' of the table. For comparison, we also use a batch algorithm [YLZ+13b] to compute the "true" SimRank scores in $G \cup \Delta G$ from scratch, as illustrated in Column 'sim_{true}'. It can be noticed that for several node-pairs (not highlighted in gray), the similarities obtained by Li *et al.*'s incremental method [LHH+10] are different from the "true" SimRank scores even if the lossless SVD is used [6] during the process of updating $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}^T$, that is, Li *et al.*'s incremental approach [LHH+10] is inherently *approximate*. In fact, as will be rigorously explained in Section 3.2, their incremental strategy may miss some eigen-information whenever $\text{rank}(\mathbf{Q}) < n$.

We also observe that the target rank r for the SVD of the matrix $\mathbf{C}$ [7] may not be chosen to be negligibly smaller than n. As an example, in Column '$\text{sim}_{\text{Li et al.}}$' of

[5] According to [JW02], the damping factor C is empirically set around 0.6–0.8, which indicates the rate of decay as similarity flows across edges.

[6] A *rank-α SVD* of the matrix $\mathbf{X} \in \mathbb{R}^{n \times n}$ is a factorization of the form $\mathbf{X}_\alpha = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$, where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n \times \alpha}$ are column-orthonormal matrices, and $\mathbf{\Sigma} \in \mathbb{R}^{\alpha \times \alpha}$ is a diagonal matrix, α is called *the target rank* of the SVD, which is given by the user.

If $\alpha = \text{rank}(\mathbf{X})$, then $\mathbf{X}_\alpha = \mathbf{X}$, and we call it the *lossless SVD* of $\mathbf{X}$.

If $\alpha < \text{rank}(\mathbf{X})$, then $\|\mathbf{X} - \mathbf{X}_\alpha\|_2$ gives the least square estimate error, and we call it the *low-rank SVD* of $\mathbf{X}$.

[7] As defined in [LHH+10], r is the target rank for the SVD of the auxiliary matrix $\mathbf{C} \triangleq \mathbf{\Sigma} + \mathbf{U}^T \cdot \Delta\mathbf{Q} \cdot \mathbf{V}$, where $\Delta\mathbf{Q}$ is the changes to $\mathbf{Q}$ for link updates.

Figure 3.1, r is chosen to be $\text{rank}(\mathbf{C}) = 9$ for the *lossless* SVD of $\mathbf{C}$. Although $r = 9$ is not negligibly smaller than $n = 15$, the accuracy of '$\mathsf{sim}_{\text{Li et al.}}$' is still undesirable as compared with '$\mathsf{sim}_{\text{true}}$', not to mention choosing $r < 9$. □

Example 3.1 tells that Li *et al.*'s incremental method [LHH+10] is approximate, and the $O(r^4n^2)$ time for updating all node-pair scores might be costly, as r is not always much smaller than n. Inspired by this, we propose a novel fast (exact[8]) algorithm for incrementally computing SimRank on link-evolving graphs. Instead of incrementally finding *the changes to the SVD of* $\mathbf{Q}$ for computing new similarities, our method can cope with the dynamic nature of link updates, by precomputing SimRank on the old entire graph once via a batch algorithm first, and then incrementally finding *SimRank updates* $\mathbf{\Delta S}$ *w.r.t.* link updates. Moreover, as links are often updated with small changes, not all node-pair similarities need to be updated. As an example in the table of Figure 3.1, many node-pair similarities (highlighted in gray) remain unchanged when edge (i, j) is added. However, it is a grand challenge to identify the "affected areas" of $\mathbf{\Delta S}$, as SimRank is defined in a recursive fashion. To resolve this problem, we formulate $\mathbf{\Delta S}$ as an aggregation of similarities based on incoming paths. There are opportunities to find its "affected areas" by detecting the changes in these paths.

3.1.2 Chapter Outlines

In this chapter, our main contributions are summarized below.

- We characterize the SimRank update matrix $\mathbf{\Delta S}$ *w.r.t.* every link update via a rank-one Sylvester matrix equation. In light of this, we devise a fast incremental algorithm that can update similarities of all n^2 node-pairs in $O(Kn^2)$ time for K iterations. (Section 3.3.1)

- We also propose an effective pruning strategy to identify the "affected areas" of $\mathbf{\Delta S}$ to skip unnecessary similarity recomputations, without loss of exactness. This

[8]Here, the "exactness" of our iterative algorithm means that it can converge to the exact SimRank solution as the number of iterations increases.

enables a further speedup in the incremental SimRank computation, which is in $O(K(nd + |\mathsf{AFF}|))$ time, where d is the average in-degree of the old graph, and $|\mathsf{AFF}|$ ($\leq n^2$) is the size of "affected areas", in practice, $|\mathsf{AFF}| \ll n^2$. (Section 3.3.2)

- We conduct extensive experiments on real and synthetic datasets to demonstrate that our algorithm (a) consistently outperforms the best known link-incremental algorithm [LHH+10], from several times to over one order of magnitude, and (b) runs much faster than the batch counterpart [YLZ+13b] when link updates are small. (Section 3.4)

The rest of the chapter is structured as follows. Section 3.2 analyzes the limitations in Li *et al.*'s incremental SimRank approach [LHH+10]. Section 3.3 introduces our incremental method for SimRank assessment. Section 3.4 presents our experimental results. Section 3.5 revisits the related work, followed by the chapter conclusion in Section 3.6.

3.2 A Fly in the Ointment in [LHH+10]

In this section, we provide theoretical analysis to show that Li *et al.*'s incremental approach [LHH+10] is *approximate* in nature, which might miss some eigen-information even if the lossless SVD is utilized for computing SimRank.

The existing incremental method [LHH+10] computes SimRank by expressing similarity matrix $\mathbf{S}$ in terms of matrices $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$, where $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ are the decomposed matrices of $\mathbf{Q}$ via SVD:

$$\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T. \tag{3.1}$$

Then, when links are changed, [LHH+10] incrementally computes the new SimRank matrix $\tilde{\mathbf{S}}$ by updating the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ as

$$\tilde{\mathbf{U}} = \mathbf{U} \cdot \mathbf{U_C}, \quad \tilde{\mathbf{\Sigma}} = \mathbf{\Sigma_C}, \quad \tilde{\mathbf{V}} = \mathbf{V} \cdot \mathbf{V_C}, \tag{3.2}$$

[9] In the sequel, we abuse a tilde to denote the updated version of a matrix, *e.g.,* $\tilde{\mathbf{U}}$ is the updated matrix of old $\mathbf{U}$ after link updates.

where $\mathbf{U_C}, \mathbf{\Sigma_C}, \mathbf{V_C}$ in Eq.(3.2) are the decomposed matrices of the auxiliary matrix $\mathbf{C} \triangleq \mathbf{\Sigma} + \mathbf{U}^T \cdot \mathbf{\Delta Q} \cdot \mathbf{V}$ via SVD, *i.e.,*

$$\mathbf{C} = \mathbf{U_C} \cdot \mathbf{\Sigma_C} \cdot \mathbf{V_C}^T, \tag{3.3}$$

and $\mathbf{\Delta Q}$ is the changes to $\mathbf{Q}$ in response to link updates.

However, in the above process, we observe that using Eq.(3.2) to update the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ may miss some eigen-information. The main problem in [LHH+10] is that the derivation of Eq.(3.2) rests on the assumption that

$$\mathbf{U} \cdot \mathbf{U}^T = \mathbf{V} \cdot \mathbf{V}^T = \mathbf{I}_n. \tag{3.4}$$

Unfortunately, Eq.(3.4) does *not* hold (unless $\mathbf{Q}$ is a full-rank matrix, *i.e.,* $\mathrm{rank}(\mathbf{Q}) = n$) because in the case of $\mathrm{rank}(\mathbf{Q}) < n$, even a "perfect" (lossless) SVD of $\mathbf{Q}$ via Eq.(3.1) would produce $n \times \alpha$ *rectangular* matrices $\mathbf{U}$ and $\mathbf{V}$ with $\alpha = \mathrm{rank}(\mathbf{Q}) < n$. Thus, $\mathrm{rank}(\mathbf{U} \cdot \mathbf{U}^T) = \alpha < n = \mathrm{rank}(\mathbf{I}_n)$, which implies that $\mathbf{U} \cdot \mathbf{U}^T \neq \mathbf{I}_n$. Similarly, $\mathbf{V} \cdot \mathbf{V}^T \neq \mathbf{I}_n$ when $\mathrm{rank}(\mathbf{Q}) < n$. Hence, Eq.(3.4) is not always true.

Example 3.2. Consider the matrix $\mathbf{Q} = \left[\begin{smallmatrix} 0 & 1 \\ 0 & 0 \end{smallmatrix}\right]$, and its lossless SVD: $\mathbf{Q} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T$ with $\mathbf{U} = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right]$, $\mathbf{\Sigma} = [1]$, $\mathbf{V} = \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right]$. One can readily verify that

$$\mathbf{U} \cdot \mathbf{U}^T = \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right] = \left[\begin{smallmatrix} 1 & 0 \\ 0 & 0 \end{smallmatrix}\right] \neq \left[\begin{smallmatrix} 1 & 0 \\ 0 & 1 \end{smallmatrix}\right] = \mathbf{I}_n \quad (n = 2),$$

whereas

$$\mathbf{U}^T \cdot \mathbf{U} = \left[\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 1 \\ 0 \end{smallmatrix}\right] = 1 = \mathbf{I}_\alpha \text{[10]} \quad (\alpha = \mathrm{rank}(\mathbf{Q}) = 1).$$

Hence, when $\mathbf{Q}$ is not full-rank, Eq.(3.4) does not always hold, but one can prove that the following identity always holds:

$$\mathbf{U}^T \cdot \mathbf{U} = \mathbf{V}^T \cdot \mathbf{V} = \mathbf{I}_\alpha$$

since the SVD ensures that $\mathbf{U}$ and $\mathbf{V}$ are *column*-orthonormal matrices, *i.e.,* every two column-vectors, say $\mathbf{x}_i$ and $\mathbf{x}_j$ of $\mathbf{U}$ (*resp.*$\mathbf{V}$) satisfy $\mathbf{x}_i{}^T \cdot \mathbf{x}_j = \begin{cases} 1, & i{=}j; \\ 0, & i{\neq}j. \end{cases}$ □

[10] The notation $\mathbf{I}_\alpha$ denotes the $\alpha \times \alpha$ identity matrix.

To clarify that Eq.(3.4) is involved in the derivation of Eq.(3.2), let us briefly recall from [LHH+10] the 4 steps for obtaining Eq.(3.2), and the problem lies in the last step.

STEP 1. Initially, when links are changed, the old $\mathbf{Q}$ is updated to new $\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q}$. Replacing $\mathbf{Q}$ by Eq.(3.1) yields

$$\tilde{\mathbf{Q}} = \mathbf{U} \cdot \mathbf{\Sigma} \cdot \mathbf{V}^T + \mathbf{\Delta Q}. \tag{3.5}$$

STEP 2. Premultiply by $\mathbf{U}^T$ and postmultiply by $\mathbf{V}$ on both sides of Eq.(3.5), and use the property $\mathbf{U}^T \cdot \mathbf{U} = \mathbf{V}^T \cdot \mathbf{V} = \mathbf{I}_\alpha$.[11] It follows that

$$\mathbf{U}^T \cdot \tilde{\mathbf{Q}} \cdot \mathbf{V} = \mathbf{\Sigma} + \mathbf{U}^T \cdot \mathbf{\Delta Q} \cdot \mathbf{V}. \tag{3.6}$$

STEP 3. Let $\mathbf{C}$ be the right-hand side of Eq.(3.6). Applying Eq.(3.3) to Eq.(3.6) yields

$$\mathbf{U}^T \cdot \tilde{\mathbf{Q}} \cdot \mathbf{V} = \mathbf{U_C} \cdot \mathbf{\Sigma_C} \cdot \mathbf{V_C}^T. \tag{3.7}$$

STEP 4. Li *et al.* [LHH+10] attempted to premultiply by $\mathbf{U}$ and postmultiply by $\mathbf{V}^T$ on both sides of Eq.(3.7) first, and then rested on the assumption of Eq.(3.4) to obtain

$$\underbrace{\mathbf{U} \cdot \mathbf{U}^T}_{\doteq \mathbf{I}_n} \cdot \tilde{\mathbf{Q}} \cdot \underbrace{\mathbf{V} \cdot \mathbf{V}^T}_{\doteq \mathbf{I}_n} = \underbrace{(\mathbf{U} \cdot \mathbf{U_C})}_{\triangleq \tilde{\mathbf{U}}} \cdot \underbrace{\mathbf{\Sigma_C}}_{\triangleq \tilde{\mathbf{\Sigma}}} \cdot \underbrace{(\mathbf{V_C}^T \cdot \mathbf{V}^T)}_{\triangleq \tilde{\mathbf{V}}^T}, \tag{3.8}$$

which is the result of Eq.(3.2).

However, the problem lies in STEP 4. As mentioned before, Eq.(3.4) does not hold when $\text{rank}(\mathbf{Q}) < n$. That is, for Eq.(3.8), $\tilde{\mathbf{Q}} \neq \tilde{\mathbf{U}} \cdot \tilde{\mathbf{\Sigma}} \cdot \tilde{\mathbf{V}}^T$. Consequently, updating the old $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ via Eq.(3.2) may produce an error (up to $\|\mathbf{I}_n - \mathbf{U} \cdot \mathbf{U}^T\|_2 = 1$, which is not practically small) in incrementally "approximating" $\tilde{\mathbf{S}}$.

Example 3.3. Consider the old $\mathbf{Q}$ and its SVD in Example 3.2. Suppose there is an added edge, associated with $\mathbf{\Delta Q} = \left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right]$.

Li *et al.* [LHH+10] first computes the auxiliary matrix $\mathbf{C}$ as

$$\mathbf{C} \triangleq \mathbf{\Sigma} + \mathbf{U}^T \cdot \mathbf{\Delta Q} \cdot \mathbf{V} = [1] + \left[\begin{smallmatrix} 1 & 0 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 0 & 0 \\ 1 & 0 \end{smallmatrix}\right] \cdot \left[\begin{smallmatrix} 0 \\ 1 \end{smallmatrix}\right] = [1].$$

[11] As mentioned in Example 3.2, since $\mathbf{U} \in \mathbb{R}^{n \times \alpha}$ is *column*-orthonormal (not *row*-orthonormal), it follows that $\mathbf{U}^T \cdot \mathbf{U} = \mathbf{I}_\alpha$, whereas $\mathbf{U} \cdot \mathbf{U}^T \neq \mathbf{I}_n$.

Then, the matrix $\mathbf{C}$ is decomposed via Eq.(3.3) into

$$\mathbf{C} = \mathbf{U_C} \cdot \mathbf{\Sigma_C} \cdot \mathbf{V_C}^T \text{ with } \mathbf{U_C} = \mathbf{\Sigma_C} = \mathbf{V_C} = [1].$$

Finally, Li *et al.* [LHH+10] update the new SVD of $\tilde{\mathbf{Q}}$ via Eq.(3.2) as

$$\tilde{\mathbf{U}} = \mathbf{U} \cdot \mathbf{U_C} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right], \quad \tilde{\mathbf{\Sigma}} = \mathbf{\Sigma_C} = [1], \quad \tilde{\mathbf{V}} = \mathbf{V} \cdot \mathbf{V_C} = \left[\begin{smallmatrix}0\\1\end{smallmatrix}\right].$$

However, one can readily verify that

$$\tilde{\mathbf{U}} \cdot \tilde{\mathbf{\Sigma}} \cdot \tilde{\mathbf{V}}^T = \left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right] \neq \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right] = \mathbf{Q} + \mathbf{\Delta Q} = \tilde{\mathbf{Q}}.$$

In comparison, a "true" SVD of $\tilde{\mathbf{Q}}$ should be

$$\tilde{\mathbf{Q}} = \hat{\mathbf{U}} \cdot \hat{\mathbf{\Sigma}} \cdot \hat{\mathbf{V}}^T \text{ with } \hat{\mathbf{U}} = \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right], \ \hat{\mathbf{\Sigma}} = \hat{\mathbf{V}} = \left[\begin{smallmatrix}1&0\\0&1\end{smallmatrix}\right].$$

This suggests that Li *et al.*'s incremental way [LHH+10] of updating $\mathbf{U}, \mathbf{\Sigma}, \mathbf{V}$ is approximate (*e.g.,* $\tilde{\mathbf{U}} = \left[\begin{smallmatrix}1\\0\end{smallmatrix}\right]$, as compared with its "true" version $\hat{\mathbf{U}} = \left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right]$, misses the eigenvector $\left[\begin{smallmatrix}0\\1\end{smallmatrix}\right]$). Worse still, the approximation error is not small in practice as $\|\tilde{\mathbf{Q}} - \tilde{\mathbf{U}} \cdot \tilde{\mathbf{\Sigma}} \cdot \tilde{\mathbf{V}}^T\|_2 = \|\left[\begin{smallmatrix}0&1\\1&0\end{smallmatrix}\right] - \left[\begin{smallmatrix}0&1\\0&0\end{smallmatrix}\right]\|_2 = 1$. □

Our analysis tells that Eq.(3.4) holds only when (i) $\mathbf{Q}$ is full-rank, and (ii) the SVD of $\mathbf{Q}$ is lossless ($n = \text{rank}(\mathbf{Q}) = \alpha$). Only in this case, Li *et al.*'s incremental method [LHH+10] produces *exact* SimRank, which does not miss any eigen-information. However, the time complexity $O(r^4n^2)$ of [LHH+10] would become $O(n^6)$, which is rather expensive. In practice, as evidenced by our statistical experiments on Stanford Large Network Dataset Collection (SNAP) [12], most real-life graphs are not full-rank, which is also in part demonstrated by our evaluations in Figure 3.3. Thus, [LHH+10] produces the approximate solution in most cases.

3.3 Our Incremental Solution

We now propose our incremental techniques for computing SimRank, with the focus on handling *unit update* (*i.e.,* a single edge insertion or deletion). Since *batch update* (*i.e.,* a

[12] http://snap.stanford.edu/data/

list of link insertions and deletions mixed together) can be decomposed into a sequence of unit updates, unit update plays a vital role in our incremental method.

The main idea of our solution is based on two methods.

(i) We first show that SimRank update matrix $\mathbf{\Delta S} \in \mathbb{R}^{n \times n}$ can be characterized as *a rank-one Sylvester matrix equation*[13]. By leveraging the rank-one structure of the matrix, we provide a novel efficient paradigm for incrementally computing $\mathbf{\Delta S}$, which only involves *matrix-vector* and *vector-vector* multiplications, as opposed to *matrix-matrix* multiplications to directly compute the new SimRank matrix $\tilde{\mathbf{S}}$.

(ii) In light of our representation of $\mathbf{\Delta S}$, we then identify the "affected areas" of $\mathbf{\Delta S}$ in response to link update $\mathbf{\Delta Q}$, and devise an effective pruning strategy to skip unnecessary similarity recomputations for link updates.

Before detailing our two methods in the subsections below, we introduce the following notations. (i) $\mathbf{e}_i$ denotes the $n \times 1$ unit vector with a 1 in the i-th entry and 0s in other entries. (ii) d_i denotes the in-degree of the node i in the old graph G.

3.3.1 Characterizing ΔS via Rank-One Sylvester Equation

We first give the big picture, followed by rigorous proofs.

Main Idea. For every edge (i, j) update, we observe that $\mathbf{\Delta Q}$ is a *rank-one* matrix, *i.e.,* there exist two column vectors $\mathbf{u}, \mathbf{v} \in \mathbb{R}^{n \times 1}$ such that $\mathbf{\Delta Q} \in \mathbb{R}^{n \times n}$ can be decomposed into the *outer product*[14] of $\mathbf{u}$ and $\mathbf{v}$ as follows:

$$\mathbf{\Delta Q} = \mathbf{u} \cdot \mathbf{v}^T.^{15} \tag{3.9}$$

Based on Eq.(3.9), we then have an opportunity to efficiently compute $\mathbf{\Delta S}$, by characterizing it as

$$\mathbf{\Delta S} = \mathbf{M} + \mathbf{M}^T, \tag{3.10}$$

[13]Given the matrices $\mathbf{A}, \mathbf{B}, \mathbf{C} \in \mathbb{R}^{n \times n}$, the Sylvester matrix equation in terms of $\mathbf{X} \in \mathbb{R}^{n \times n}$ takes the form: $\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C}$. When $\mathbf{C}$ is a rank-α ($\leq n$) matrix, we call it *the rank-α Sylvester equation.*

[14]The *outer product* of the column vectors $\mathbf{x}, \mathbf{y} \in \mathbb{R}^{n \times 1}$ is an $n \times n$ rank-1 matrix $\mathbf{x} \cdot \mathbf{y}^T$, in contrast with the *inner product* $\mathbf{x}^T \cdot \mathbf{y}$, which is a scalar.

[15]The explict expression of $\mathbf{u}$ and $\mathbf{v}$ will be given after a few discussions.

where the auxiliary matrix $\mathbf{M} \in \mathbb{R}^{n \times n}$ satisfies the following *rank-one* Sylvester equation:

$$\mathbf{M} = C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{M} \cdot \tilde{\mathbf{Q}}^T + C \cdot \mathbf{u} \cdot \mathbf{w}^T. \tag{3.11}$$

Here, $\mathbf{u}, \mathbf{w}$ are two column vectors: $\mathbf{u}$ is derived from Eq.(3.9), and $\mathbf{w}$ can be represented in terms of the old $\mathbf{Q}$ and $\mathbf{S}$ (we will provide their exact expressions later after some discussions); and $\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{\Delta Q}$.

Thus, computing $\mathbf{\Delta S}$ boils down to solving $\mathbf{M}$ in Eq.(3.11). The main advantage of solving $\mathbf{M}$ via Eq.(3.11), as compared to directly computing the new scores $\tilde{\mathbf{S}}$ via SimRank formula

$$\tilde{\mathbf{S}} = C \cdot \tilde{\mathbf{Q}} \cdot \tilde{\mathbf{S}} \cdot \tilde{\mathbf{Q}}^T + (1 - C) \cdot \mathbf{I}_n, \tag{3.12}$$

is the high computational efficiency. More specifically, solving $\tilde{\mathbf{S}}$ via Eq.(3.12) needs expensive *matrix-matrix* multiplications, whereas computing $\mathbf{M}$ via Eq.(3.11) involves only *matrix-vector* and *vector-vector* multiplications, which is a substantial improvement achieved by our observation that $(C \cdot \mathbf{u}\mathbf{w}^T) \in \mathbb{R}^{n \times n}$ in Eq.(3.11) is a *rank*-1 matrix, as opposed to the (full) *rank-n* matrix $(1 - C) \cdot \mathbf{I}_n$ in Eq.(3.12). To further elaborate on this, we readily convert the recursive forms of Eqs.(3.11) and (3.12), respectively, into the following series forms: [16]

$$\mathbf{M} = \sum_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{u} \cdot \mathbf{w}^T \cdot (\tilde{\mathbf{Q}}^T)^k, \tag{3.13}$$

$$\tilde{\mathbf{S}} = (1 - C) \cdot \sum_{k=0}^{\infty} C^k \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{I}_n \cdot (\tilde{\mathbf{Q}}^T)^k. \tag{3.14}$$

To compute the sums in Eq.(3.13) for $\mathbf{M}$, a conventional way is to memoize $\mathbf{M}_0 \leftarrow C \cdot \mathbf{u} \cdot \mathbf{w}^T$ first (where the intermediate result $\mathbf{M}_0$ is an $n \times n$ matrix), and then iterate as follows:

$$\mathbf{M}_{k+1} \leftarrow \mathbf{M}_0 + C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{M}_k \cdot \tilde{\mathbf{Q}}^T, \quad (k = 0, 1, 2, \cdots)$$

involving costly *matrix-matrix* multiplications (*e.g.*, $\tilde{\mathbf{Q}} \cdot \mathbf{M}_k$). In contrast, our trick takes advantage of the *rank-one* structure of $\mathbf{u} \cdot \mathbf{w}^T$ to compute the sums in Eq.(3.13) for $\mathbf{M}$,

[16] One can readily verify that if $\mathbf{X} = \sum_{k=0}^{\infty} \mathbf{A}^k \cdot \mathbf{C} \cdot \mathbf{B}^k$ is a convergent matrix series, it is the solution of the Sylvester equation $\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C}$.

by converting the conventional *matrix-matrix* multiplications $\tilde{\mathbf{Q}} \cdot (\mathbf{u}\mathbf{w}^T) \cdot \tilde{\mathbf{Q}}^T$ into *matrix-vector* and *vector-vector* operations $(\tilde{\mathbf{Q}}\mathbf{u}) \cdot (\tilde{\mathbf{Q}}\mathbf{w})^T$. More specifically, by leveraging two auxiliary vectors $\boldsymbol{\xi}_k, \boldsymbol{\eta}_k$, we adopt the following iterative paradigm to compute Eq.(3.13):

1. initialize $\boldsymbol{\xi}_0 \leftarrow C \cdot \mathbf{u}, \quad \boldsymbol{\eta}_0 \leftarrow \mathbf{w}, \quad \mathbf{M}_0 \leftarrow C \cdot \mathbf{u} \cdot \mathbf{w}^T$
2. for $k = 0, 1, 2, \cdots$
3. $\quad \boldsymbol{\xi}_{k+1} \leftarrow C \cdot \tilde{\mathbf{Q}} \cdot \boldsymbol{\xi}_k, \quad \boldsymbol{\eta}_{k+1} \leftarrow \tilde{\mathbf{Q}} \cdot \boldsymbol{\eta}_k$
4. $\quad \mathbf{M}_{k+1} \leftarrow \boldsymbol{\xi}_{k+1} \cdot \boldsymbol{\eta}_{k+1}^T + \mathbf{M}_k$

which only requires *matrix-vector* multiplications (*e.g.*, $\tilde{\mathbf{Q}} \cdot \boldsymbol{\xi}_k$) and *vector-vector* multiplications (*e.g.*, $\boldsymbol{\xi}_{k+1} \cdot \boldsymbol{\eta}_{k+1}^T$), without the need to perform *matrix-matrix* multiplications.

It is worth mentioning that our above trick is solely suitable for efficiently computing $\mathbf{M}$ in Eq.(3.13), but not applicable to accelerating $\tilde{\mathbf{S}}$ computation in Eq.(3.14). This is because $\mathbf{I}_n$ is a (full) rank-n matrix that cannot be decomposed into the outer product of two vectors. Thus, our trick is particularly tailored for improving the *incremental* computation of $\Delta\mathbf{S}$ via Eq.(3.11), rather than the *batch* computation of $\tilde{\mathbf{S}}$ via Eq.(3.12).

Finding $\mathbf{u}, \mathbf{v}, \mathbf{w}$ for Eqs.(3.9) and (3.11). The challenging tasks in characterizing $\Delta\mathbf{S}$ for our incremental method are (i) to find the vectors $\mathbf{u}, \mathbf{v}$ in Eq.(3.9) for the *rank-one* decomposition of $\Delta\mathbf{Q}$, and (ii) to express the vector $\mathbf{w}$ in Eq.(3.11) in terms of the old matrices $\mathbf{Q}$ and $\mathbf{S}$ for guaranteeing that Eq.(3.11) is a *rank-one* Sylvester equation.

To find $\mathbf{u}$ and $\mathbf{v}$ in Eq.(3.9), we show the following theorem.

Theorem 3.4. *If there is an edge (i, j) inserted into G, then the change in $\mathbf{Q}$ is an $n \times n$ rank-one matrix, i.e., $\Delta\mathbf{Q} = \mathbf{u} \cdot \mathbf{v}^T$, where*

$$\mathbf{u} = \begin{cases} \mathbf{e}_j & (d_j = 0) \\ \frac{1}{d_j+1}\mathbf{e}_j & (d_j > 0) \end{cases}, \quad \mathbf{v} = \begin{cases} \mathbf{e}_i & (d_j = 0) \\ \mathbf{e}_i - [\mathbf{Q}]_{j,\star}^T & (d_j > 0) \end{cases} \tag{3.15}$$

If there is an edge (i, j) deleted from G, then the change in $\mathbf{Q}$ can be decomposed as $\Delta\mathbf{Q} = \mathbf{u} \cdot \mathbf{v}^T$, where

$$\mathbf{u} = \begin{cases} \mathbf{e}_j & (d_j = 1) \\ \frac{1}{d_j-1}\mathbf{e}_j & (d_j > 1) \end{cases}, \quad \mathbf{v} = \begin{cases} -\mathbf{e}_i & (d_j = 1) \\ [\mathbf{Q}]_{j,\star}^T - \mathbf{e}_i & (d_j > 1) \end{cases} \quad \square \tag{3.16}$$

Proof. Due to space limitations, we shall only prove the insertion case. A similar proof holds for the deletion case.

(i) If $d_j = 0$, $[\mathbf{Q}]_{j,\star} = \mathbf{0}$. Thus, for the inserted edge (i, j), $[\mathbf{Q}]_{j,i}$ will be updated from 0 to 1, *i.e.,* $\mathbf{\Delta Q} = \mathbf{e}_j \mathbf{e}_i^T$.

(ii) If $d_j > 0$, all the nonzero entries in $[\mathbf{Q}]_{j,\star}$ are $\frac{1}{d_j}$. Thus, for the inserted edge (i, j), the old $\mathbf{Q}$ can be converted into the new $\tilde{\mathbf{Q}}$ via 2 steps, as depicted below:

$$\mathbf{Q} = \begin{bmatrix} \cdots & & \cdots & & \cdots & & \cdots \\ \cdots & \frac{1}{d_j} & \cdots & 0 & \cdots & \frac{1}{d_j} & \cdots \\ \cdots & & \cdots & & \cdots & & \cdots \end{bmatrix} \begin{matrix} \\ (j\text{-th row}) \\ \\ \end{matrix} \quad (i\text{-th col})$$

$$\xrightarrow{\frac{d_j}{d_j+1} \times (j\text{-th row})} \begin{bmatrix} \cdots & & \cdots & \cdots & & \cdots \\ \cdots & \boxed{\frac{1}{d_j+1}} & \cdots & 0 \cdots & \boxed{\frac{1}{d_j+1}} & \cdots \\ \cdots & & \cdots & \cdots & & \cdots \end{bmatrix} (j\text{-th row})$$

$$\xrightarrow{\frac{1}{d_j+1} + (j,i)\text{-entry}} \begin{bmatrix} \cdots & & \cdots & & \cdots & & \cdots \\ \cdots & \frac{1}{d_j+1} & \cdots & \boxed{\frac{1}{d_j+1}} & \cdots & \frac{1}{d_j+1} & \cdots \\ \cdots & & \cdots & & \cdots & & \cdots \end{bmatrix} = \tilde{\mathbf{Q}}$$

(a) We change all nonzero entries of $[\mathbf{Q}]_{j,\star}$ from $\frac{1}{d_j}$ to $\frac{1}{d_j+1}$, by multiplying $\frac{d_j}{d_j+1}$ on the j-th row of $\mathbf{Q}$. Recall from *the elementary matrix property* that multiplying the j-th row of a matrix by $\alpha \neq 0$ can be accomplished by using $\mathbf{I} - (1 - \alpha)\mathbf{e}_j \mathbf{e}_j^T$ as a left-hand multiplier on the matrix. Hence, after this step, $\mathbf{Q}$ is converted into the matrix $\mathbf{Q}'$, *i.e.,*

$$\mathbf{Q}' = (\mathbf{I} - (1 - \tfrac{d_j}{d_j+1})\mathbf{e}_j \mathbf{e}_j^T) \cdot \mathbf{Q} = \mathbf{Q} - \tfrac{1}{d_j+1} \mathbf{e}_j \cdot [\mathbf{Q}]_{j,\star}.$$

(b) We next update the (j, i)-entry of $\mathbf{Q}'$ from 0 to $\frac{1}{d_j+1}$, which yields the new $\tilde{\mathbf{Q}}$, *i.e.,*

$$\tilde{\mathbf{Q}} = \mathbf{Q}' + \tfrac{1}{d_j+1} \mathbf{e}_j \mathbf{e}_i^T = \mathbf{Q} - \tfrac{1}{d_j+1} \mathbf{e}_j \cdot ([\mathbf{Q}]_{j,\star} - \mathbf{e}_i^T).$$

Since $\mathbf{\Delta Q} = \tilde{\mathbf{Q}} - \mathbf{Q}$, it follows that

$$\mathbf{\Delta Q} = \mathbf{u} \cdot \mathbf{v}^T, \quad \text{with } \mathbf{u} := \tfrac{1}{d_j+1}\mathbf{e}_j, \quad \mathbf{v}^T := (\mathbf{e}_i^T - [\mathbf{Q}]_{j,\star}).$$

which proves the case $d_j > 0$ in Eq.(3.15). □

Example 3.5. Consider the graph G in Figure 3.1. Suppose there is an edge (i, j) inserted into G. As in the old G, $d_j = 2 > 0$ and

$$[\mathbf{Q}]_{j,\star} = \Big[\, 0 \cdots 0 \overset{(h)}{\tfrac{1}{2}} 0\ 0 \overset{(k)}{\tfrac{1}{2}} 0 \cdots 0 \,\Big] \in \mathbb{R}^{1\times 15},$$

according to Theorem 3.4, the change in $\mathbf{Q}$ is a 15×15 rank-one matrix, which can be decomposed as $\mathbf{\Delta Q} = \mathbf{u} \cdot \mathbf{v}^T$ with $\mathbf{u} = \frac{1}{d_j+1}\mathbf{e}_j = \frac{1}{3}\mathbf{e}_j = \Big[\, 0 \cdots 0 \overset{(j)}{\tfrac{1}{3}} 0 \cdots 0 \,\Big]^T \in \mathbb{R}^{15\times 1}$,

$$\mathbf{v} = \mathbf{e}_i - [\mathbf{Q}]_{j,\star}^T = \Big[\, 0 \cdots 0 \overset{(h)}{-\tfrac{1}{2}} \overset{(i)}{1} \overset{(j)}{0} \overset{(k)}{-\tfrac{1}{2}} 0 \cdots 0 \,\Big]^T \in \mathbb{R}^{15\times 1}.$$ □

For every link update, Theorem 3.4 suggests that the change $\mathbf{\Delta Q}$ has a very special structure — the $n \times n$ *rank-one* matrix. More importantly, it finds a rank-one decomposition for $\mathbf{\Delta Q}$, by expressing the vectors $\mathbf{u}$ and $\mathbf{v}$ in terms of d_j and $[\mathbf{Q}]_{j,\star}^T$. It should be noted that such a rank-one decomposition is not unique, since for any scalar $\lambda \neq 0$, the vectors $\mathbf{u}' \triangleq \lambda \cdot \mathbf{u}$ and $\mathbf{v}' \triangleq \frac{\mathbf{v}}{\lambda}$ can be another rank-one decomposition for $\mathbf{\Delta Q}$. However, for any $\mathbf{u}$ and $\mathbf{v}$ that satisfy Eq.(3.9), there exists a vector $\mathbf{w}$ such that Eq.(3.11) is a *rank-one* Sylvester equation.

Capitalizing on Theorem 3.4, we are now ready to determine the expression of $\mathbf{w}$ in Eq.(3.11) in terms of the old $\mathbf{Q}$ and $\mathbf{S}$.

Theorem 3.6. *Suppose there is an edge (i, j) updated in G. Let $\mathbf{u}$ and $\mathbf{v}$ be the rank-one decomposition of $\mathbf{\Delta Q}$ in Theorem 3.4. Then, (i) there exists a vector $\mathbf{w} = \mathbf{y} + \frac{\lambda}{2}\mathbf{u}$ with*

$$\mathbf{y} = \mathbf{Q} \cdot \mathbf{z}, \quad \lambda = \mathbf{v}^T \cdot \mathbf{z}, \quad \mathbf{z} = \mathbf{S} \cdot \mathbf{v} \tag{3.17}$$

such that Eq.(3.11) is the rank-one *Sylvester equation.*

(ii) Utilizing the solution $\mathbf{M}$ *to Eq.*(3.11)*, the SimRank update matrix* $\Delta\mathbf{S}$ *can be represented by Eq.*(3.10). □

Proof. We show this by following the two steps:

(a) We find a recursion for the SimRank update matrix $\Delta\mathbf{S}$.

To characterize $\Delta\mathbf{S}$ in terms of the old $\mathbf{Q}$ and $\mathbf{S}$, we subtract Eq.(2.2) from Eq.(3.12), and apply $\Delta\mathbf{S} = \tilde{\mathbf{S}} - \mathbf{S}$, yielding

$$\Delta\mathbf{S} = C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{S} \cdot \tilde{\mathbf{Q}}^T + C \cdot \tilde{\mathbf{Q}} \cdot \Delta\mathbf{S} \cdot \tilde{\mathbf{Q}}^T - C \cdot \mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T. \tag{3.18}$$

By Theorem 3.4, there exist two vectors $\mathbf{u}$ and $\mathbf{v}$ such that

$$\tilde{\mathbf{Q}} = \mathbf{Q} + \Delta\mathbf{Q} = \mathbf{Q} + \mathbf{u} \cdot \mathbf{v}^T. \tag{3.19}$$

Then, we plug Eq.(3.19) into the term $C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{S} \cdot \tilde{\mathbf{Q}}^T$ of Eq.(3.18), and simplify the result into

$$\Delta\mathbf{S} = C \cdot \tilde{\mathbf{Q}} \cdot \Delta\mathbf{S} \cdot \tilde{\mathbf{Q}}^T + C \cdot \mathbf{T} \tag{3.20}$$

$$\text{with } \mathbf{T} = \mathbf{u}(\mathbf{Q}\mathbf{S}\mathbf{v})^T + (\mathbf{Q}\mathbf{S}\mathbf{v})\mathbf{u}^T + (\mathbf{v}^\mathbf{T}\mathbf{S}\mathbf{v})\mathbf{u}\mathbf{u}^T. \tag{3.21}$$

We can readily verify that matrix $\mathbf{T}$ is symmetric ($\mathbf{T} = \mathbf{T}^T$). Moreover, we note that $\mathbf{T}$ is the sum of two rank-one matrices. This can be verified by letting $\mathbf{z} \triangleq \mathbf{S} \cdot \mathbf{v}$, $\mathbf{y} \triangleq \mathbf{Q} \cdot \mathbf{z}$, $\lambda \triangleq \mathbf{v}^T \cdot \mathbf{z}$.

Then, utilizing the auxiliary vectors $\mathbf{z}, \mathbf{y}$ and the scalar λ, Eq.(3.21) can be simplified into the following form:

$$\mathbf{T} = \mathbf{u} \cdot \mathbf{w}^T + \mathbf{w} \cdot \mathbf{u}^T, \quad \text{with } \mathbf{w} = \mathbf{y} + \tfrac{\lambda}{2}\mathbf{u}. \tag{3.22}$$

(b) We next convert the recursion of $\Delta\mathbf{S}$ into the series form.

One can readily verify that the solution $\mathbf{X}$ to the matrix equation $\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C}$ has the following closed form:

$$\mathbf{X} = \mathbf{A} \cdot \mathbf{X} \cdot \mathbf{B} + \mathbf{C} \quad \Leftrightarrow \quad \mathbf{X} = \sum_{k=0}^{\infty} \mathbf{A}^k \cdot \mathbf{C} \cdot \mathbf{B}^k \tag{3.23}$$

Thus, based on Eq.(3.23), the recursive definition of $\mathbf{\Delta S}$ in Eq.(3.20) naturally leads itself to the following series form:

$$\mathbf{\Delta S} = \sum_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{T} \cdot (\tilde{\mathbf{Q}}^T)^k.$$

Combining this with Eq.(3.22) yields

$$\begin{aligned} \mathbf{\Delta S} &= \sum_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \left(\mathbf{u} \cdot \mathbf{w}^T + \mathbf{w} \cdot \mathbf{u}^T\right) \cdot (\tilde{\mathbf{Q}}^T)^k \\ &= \mathbf{M} + \mathbf{M}^T \quad \text{with } \mathbf{M} \text{ being defined in Eq.(3.13).} \end{aligned}$$

In light of Eq.(3.23), the series form of $\mathbf{M}$ in Eq.(3.13) satisfies the rank-one Sylvester recursive form of Eq.(3.11). □

Theorem 3.6 obtains an exact expression for $\mathbf{w}$ in Eq.(3.11). To be precise, given $\mathbf{Q}$ and $\mathbf{S}$ in the old graph G, and an edge (i, j) updated to G, one can find $\mathbf{u}$ and $\mathbf{v}$ via Theorem 3.4 first, and then resort to Theorem 3.6 to compute $\mathbf{w}$ from $\mathbf{u}, \mathbf{v}, \mathbf{Q}, \mathbf{S}$. Because of the existence of the vector $\mathbf{w}$, the Sylvester form of Eq.(3.11) being *rank-one* can be guaranteed. Henceforth, our aforementioned trick can be deployed to iteratively compute $\mathbf{M}$ in Eq.(3.13), needing no *matrix-matrix* multiplications.

Computing $\mathbf{\Delta S}$. Determining $\mathbf{w}$ via Theorem 3.6 is intended to speed up the incremental computation of $\mathbf{\Delta S}$. Indeed, for each link update, the *whole* process of computing $\mathbf{\Delta S}$ in Eq.(3.10), given $\mathbf{Q}$ and $\mathbf{S}$, needs no *matrix-matrix* multiplications at all. Specifically, the computation of $\mathbf{\Delta S}$ consists of two phases: (i) Given $\mathbf{Q}$ and $\mathbf{S}$, we compute $\mathbf{w}$ via Theorems 3.4 and 3.6. This phase merely includes the matrix-vector multiplications (*e.g.,* $\mathbf{Qz}, \mathbf{Sv}$), the inner product of vectors (*e.g.,* $\mathbf{v}^T\mathbf{z}$), and the vector scaling and additions, *i.e.,* SAXPY (*e.g.,* $\mathbf{y} + \frac{\lambda}{2}\mathbf{u}$). (ii) Given $\mathbf{w}$, we compute $\mathbf{M}$ via Eq.(3.13). In this phase, our novel iterative paradigm for Eq.(3.13), as mentioned earlier, can circumvent the *matrix-matrix* multiplications. Thus, taking (i) and (ii) together, it suffices to harness only *matrix-vector* and *vector-vector* operations in whole process of computing $\mathbf{\Delta S}$.

Leveraging Theorems 3.4 and 3.6, we are able to characterize the SimRank change $\mathbf{\Delta S}$, based on the following theorem.

Theorem 3.7. *When there is an edge (i,j) updated in G, then the SimRank change $\Delta\mathbf{S}$ can be characterized as*

$$\Delta\mathbf{S} = \mathbf{M} + \mathbf{M}^T \quad \textit{with}$$
$$\mathbf{M} = \sum_{k=0}^{\infty} C^{k+1} \cdot \tilde{\mathbf{Q}}^k \cdot \mathbf{e}_j \cdot \boldsymbol{\gamma}^T \cdot (\tilde{\mathbf{Q}}^T)^k, \tag{3.24}$$

where the auxiliary vector $\boldsymbol{\gamma}$ is obtained as follows:

(i) For the edge insertion, $\boldsymbol{\gamma} =$

$$\begin{cases} \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j & (d_j = 0) \\ \frac{1}{(d_j+1)} \left(\mathbf{Q} \cdot [\mathbf{S}]_{\star,i} - \frac{1}{C} \cdot [\mathbf{S}]_{\star,j} + (\frac{\lambda}{2(d_j+1)} + \frac{1}{C} - 1) \cdot \mathbf{e}_j \right) & (d_j > 0) \end{cases} \tag{3.25}$$

(ii) For the edge deletion, $\boldsymbol{\gamma} =$

$$\begin{cases} -\mathbf{Q} \cdot [\mathbf{S}]_{\star,i} + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j & (d_j = 1) \\ \frac{1}{(d_j-1)} \left(\frac{1}{C} \cdot [\mathbf{S}]_{\star,j} - \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} + (\frac{\lambda}{2(d_j-1)} - \frac{1}{C} + 1) \cdot \mathbf{e}_j \right) & (d_j > 1) \end{cases} \tag{3.26}$$

and the scalar λ can be derived from

$$\lambda = [\mathbf{S}]_{i,i} + \frac{1}{C} \cdot [\mathbf{S}]_{j,j} - 2 \cdot [\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i} - \frac{1}{C} + 1. \quad \square \tag{3.27}$$

Proof. For space interests, we merely show insertion case.

(i) When $d_j = 0$, by Eq.(3.15) in Theorem 3.4, $\mathbf{v} = \mathbf{e}_i$, $\mathbf{u} = \mathbf{e}_j$. Plugging them into Eq.(3.17) gets $\mathbf{z} = [\mathbf{S}]_{\star,i}$, $\mathbf{y} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i}$, $\lambda = [\mathbf{S}]_{i,i}$. Thus, by virtue of $\mathbf{w} = \mathbf{y} + \frac{\lambda}{2}\mathbf{u}$ in Theorem 3.6, we have $\mathbf{w} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j$. Coupling this with Eq.(3.13), $\mathbf{u} = \mathbf{e}_j$, and Theorem 3.6 proves the case $d_j = 0$ in Eq.(3.25).

(ii) When $d_j > 0$, Eq.(3.15) in Theorem 3.4 indicates that

$$\mathbf{v} = \mathbf{e}_i - [\mathbf{Q}]^T_{j,\star}, \quad \mathbf{u} = \tfrac{1}{d_j+1} \cdot \mathbf{e}_j. \tag{3.28}$$

Substituting these back into Eq.(3.17) yields

$$\mathbf{z} = [\mathbf{S}]_{\star,i} - \mathbf{S} \cdot [\mathbf{Q}]^T_{j,\star}, \quad \mathbf{y} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} - \mathbf{Q} \cdot \mathbf{S} \cdot [\mathbf{Q}]^T_{j,\star},$$
$$\lambda = [\mathbf{S}]_{i,i} - 2 \cdot [\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i} + [\mathbf{Q}]_{j,\star} \cdot \mathbf{S} \cdot [\mathbf{Q}]^T_{j,\star}.$$

To simplify $\mathbf{Q} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T$ in $\mathbf{y}$, and $[\mathbf{Q}]_{j,\star} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T$ in λ, we postmultiply both sides of Eq.(2.2) by $\mathbf{e}_j$ to obtain

$$\mathbf{Q} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T = \tfrac{1}{C} \cdot ([\mathbf{S}]_{\star,j} - (1-C) \cdot \mathbf{e}_j). \tag{3.29}$$

We also premultiply both sides of Eq.(3.29) by $\mathbf{e}_j^T$ to get

$$[\mathbf{Q}]_{j,\star} \cdot \mathbf{S} \cdot [\mathbf{Q}]_{j,\star}^T = \tfrac{1}{C} \cdot ([\mathbf{S}]_{j,j} - 1) + 1. \tag{3.30}$$

Plugging Eqs.(3.29) and (3.30) into $\mathbf{y}$ and λ, respectively, and then plugging the resulting $\mathbf{y}$ and λ into $\mathbf{w} = \mathbf{y} + \frac{\lambda}{2}\mathbf{u}$ produce

$$\mathbf{w} = \mathbf{Q} \cdot [\mathbf{S}]_{\star,i} - \tfrac{1}{C} \cdot [\mathbf{S}]_{\star,j} + (\tfrac{1}{C} + \tfrac{\lambda}{2(d_j+1)} - 1) \cdot \mathbf{e}_j,$$

with $\lambda = [\mathbf{S}]_{i,i} + \frac{1}{C} \cdot [\mathbf{S}]_{j,j} - 2 \cdot [\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i} - \frac{1}{C} + 1$.

Combining this with Eqs.(3.13), (3.28) shows the case $d_j > 0$ for Eq.(3.25). Finally, taking (i) and (ii) together with Theorem 3.6 completes the proof for the link insertion case. □

For each link update, Theorem 3.7 provides a novel method to compute the incremental SimRank matrix $\Delta\mathbf{S}$, by utilizing the previous information of $\mathbf{Q}$ and $\mathbf{S}$ in the original graph G, as opposed to [LHH+10] that entails the incremental SVD maintenance. To *efficiently* compute $\Delta\mathbf{S}$ via Theorem 3.7, two tricks are worth mentioning. (i) We observe that, by viewing the matrix $\mathbf{Q}$ as a stack of row vectors, the j-th row of the term $(\mathbf{Q} \cdot [\mathbf{S}]_{\star,i})$ in Eqs.(3.25) and (3.26) is actually the inner product $[\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i}$, being the term in Eq.(3.27). Thus, the resulting $[\mathbf{Q} \cdot [\mathbf{S}]_{\star,i}]_{j,\star}$, once computed, can be reused to compute $[\mathbf{Q}]_{j,\star} \cdot [\mathbf{S}]_{\star,i}$ in λ. (ii) As suggested earlier, computing the matrix series for $\mathbf{M}$ needs no matrix-matrix multiplications at all, but involves the matrix-vector multiplications iteratively (*e.g.*, $\boldsymbol{\eta}_{k+1} \leftarrow \tilde{\mathbf{Q}} \cdot \boldsymbol{\eta}_k$). Since $\tilde{\mathbf{Q}} = \mathbf{Q} + \mathbf{u} \cdot \mathbf{v}^T$ via Theorem 3.4, we notice that $\tilde{\mathbf{Q}} \cdot \boldsymbol{\eta}_k$ can be computed more efficiently, with no need to memoize $\tilde{\mathbf{Q}}$ in extra memory space, as follows: $\tilde{\mathbf{Q}} \cdot \boldsymbol{\eta}_k = \mathbf{Q} \cdot \boldsymbol{\eta}_k + (\mathbf{v}^T \cdot \boldsymbol{\eta}_k) \cdot \mathbf{u}$.

Algorithm. Based on Theorem 3.7, we provide an incremental SimRank algorithm, denoted as Inc-uSR, for each link update.

Algorithm 3.1: Inc-uSR $(G, \mathbf{S}, K, (i,j), C)$

Input : a graph G, old similarities $\mathbf{S}$ for G, #-iteration K, the edge (i,j) updated to G, and damping factor C.

Output: the new similarities $\tilde{\mathbf{S}}$ for $G \cup \{(i,j)\}$.

1 initialize the transition matrix $\mathbf{Q}$ in G ;

2 $d_j :=$ in-degree of node j in G ;

3 memoize $\mathbf{w} := \mathbf{Q} \cdot [\mathbf{S}]_{\star,i}$;

4 compute $\lambda := [\mathbf{S}]_{i,i} + \frac{1}{C} \cdot [\mathbf{S}]_{j,j} - 2 \cdot [\mathbf{w}]_j - \frac{1}{C} + 1$;

5 **if** *edge* (i,j) *is to be inserted* **then**

6 $\quad$ **if** $d_j = 0$ **then** $\mathbf{u} := \mathbf{e}_j,\ \mathbf{v} := \mathbf{e}_i,\ \boldsymbol{\gamma} := \mathbf{w} + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j$;

7 $\quad$ **else** $\mathbf{u} := \frac{1}{d_j+1}\mathbf{e}_j, \quad \mathbf{v} := \mathbf{e}_i - [\mathbf{Q}]^T_{j,\star}$;

8 $\quad\quad \boldsymbol{\gamma} := \frac{1}{(d_j+1)}\big(\mathbf{w} - \frac{1}{C}[\mathbf{S}]_{\star,j} + (\frac{\lambda}{2(d_j+1)} + \frac{1}{C} - 1)\mathbf{e}_j\big)$;

9 **else if** *edge* (i,j) *is to be deleted* **then**

10 $\quad$ **if** $d_j = 1$ **then** $\mathbf{u} := \mathbf{e}_j,\ \mathbf{v} := -\mathbf{e}_i,\ \boldsymbol{\gamma} := \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j - \mathbf{w}$;

11 $\quad$ **else** $\mathbf{u} := \frac{1}{d_j-1}\mathbf{e}_j, \quad \mathbf{v} := [\mathbf{Q}]^T_{j,\star} - \mathbf{e}_i$;

12 $\quad\quad \boldsymbol{\gamma} := \frac{1}{(d_j-1)}\big(\frac{1}{C}[\mathbf{S}]_{\star,j} - \mathbf{w} + (\frac{\lambda}{2(d_j-1)} - \frac{1}{C} + 1)\mathbf{e}_j\big)$;

13 initialize $\boldsymbol{\xi}_0 := C \cdot \mathbf{e}_j, \quad \boldsymbol{\eta}_0 := \boldsymbol{\gamma}, \quad \mathbf{M}_0 := C \cdot \mathbf{e}_j \cdot \boldsymbol{\gamma}^T$;

14 **for** $k = 0, 1, \cdots, K-1$ **do**

15 $\quad \boldsymbol{\xi}_{k+1} := C \cdot \mathbf{Q} \cdot \boldsymbol{\xi}_k + C \cdot (\mathbf{v}^T \cdot \boldsymbol{\xi}_k) \cdot \mathbf{u}$;

16 $\quad \boldsymbol{\eta}_{k+1} := \mathbf{Q} \cdot \boldsymbol{\eta}_k + (\mathbf{v}^T \cdot \boldsymbol{\eta}_k) \cdot \mathbf{u}$;

17 $\quad \mathbf{M}_{k+1} := \boldsymbol{\xi}_{k+1} \cdot \boldsymbol{\eta}^T_{k+1} + \mathbf{M}_k$;

18 $\tilde{\mathbf{S}} := \mathbf{S} + \mathbf{M}_K + \mathbf{M}^T_K$;

19 **return** $\tilde{\mathbf{S}}$;

Given the old graph G, the old similarities $\mathbf{S}$ in G, the edge (i,j) updated to G, and the damping factor C, the algorithm incrementally computes the new similarities $\tilde{\mathbf{S}}$ in $G \cup \{(i,j)\}$. It works as follows. First, it initializes the transition matrix $\mathbf{Q}$ and in-degree d_j of node j in G (lines 1–2). Using $\mathbf{Q}$ and $\mathbf{S}$, it precomputes the auxiliary

vector $\mathbf{w}$ and scalar λ (lines 3–4). Once computed, both $\mathbf{w}$ and λ are memoized for precomputing (i) vectors $\mathbf{u}$ and $\mathbf{v}$ for a rank-one factorization of $\Delta\mathbf{Q}$, and (ii) initial vector $\boldsymbol{\gamma}$ for subsequent $\mathbf{M}_k$ iterations (lines 5–12). Then, the algorithm maintains two auxiliary vectors $\boldsymbol{\xi}_k$ and $\boldsymbol{\eta}_k$ to iteratively compute matrix $\mathbf{M}_k$ (lines 13–17). The process continues until the number of iterations reaches a given K. Finally, the new scores $\tilde{\mathbf{S}}$ are obtained by $\mathbf{M}_K$[17] (line 18).

Example 3.8. Recall the old graph G and $\mathbf{S}$ of G from Figure 3.1. When edge (i, j) is added, we show how Inc-uSR computes the new $\tilde{\mathbf{S}}$, which is in part depicted in Column '$\mathsf{sim}_{\mathsf{true}}$'.

Given the following information from the old $\mathbf{S}$ below: [18]

$$[\mathbf{S}]_{\star,i} = \Big[0,\ \cdots,\ 0,\ \overset{(f)}{0.246},\ \overset{(g)}{0},\ \overset{(h)}{0},\ \overset{(i)}{0.590},\ \overset{(j)}{0.310},\ 0,\ \cdots,\ 0\Big]^T \in \mathbb{R}^{15\times 1},$$

$$[\mathbf{S}]_{\star,j} = \Big[0,\ \cdots,\ 0,\ \overset{(f)}{0.246},\ \overset{(g)}{0},\ \overset{(h)}{0},\ \overset{(i)}{0.310},\ \overset{(j)}{0.510},\ 0,\ \cdots,\ 0\Big]^T \in \mathbb{R}^{15\times 1},$$

as $d_j = 2$, Inc-uSR first precomputes $\mathbf{w}$ and λ via lines 3–4:

$$\mathbf{w} = \Big[\overset{(a)}{0.104},\ \overset{(b)}{0.139},\ 0,\ \cdots,\ 0\Big]^T \in \mathbb{R}^{15\times 1},$$
$$\lambda = 0.590 + \frac{1}{0.8}\times 0.510 - 2\times 0 - \frac{1}{0.8} + 1 = 0.978.$$

As an "edge insertion" operation, the vectors $\mathbf{u}$ and $\mathbf{v}$ for a rank-one decomposition of $\Delta\mathbf{Q}$ can be computed via line 7. Their results are depicted in Example 3.5.

Utilizing $\mathbf{w}$ and λ, the vector $\boldsymbol{\gamma}$ can be obtained via line 8:

$$\boldsymbol{\gamma} = \frac{1}{(2+1)}\times\Big(\mathbf{w} - \frac{1}{0.8}[\mathbf{S}]_{\star,j} + \big(\frac{\lambda}{2\times(2+1)} + \frac{1}{0.8} - 1\big)\mathbf{e}_j\Big)$$

$$= \Big[\overset{(a)}{0.035},\ \overset{(b)}{0.046},\ 0,\ 0,\ 0,\ \overset{(f)}{-0.086}\ 0,\ 0,\ \overset{(i)}{-0.129},\ \overset{(j)}{-0.075},\ 0,\ \cdots,\ 0\Big]^T \in \mathbb{R}^{15\times 1}$$

[17] It can be proved that $\|\mathbf{M}_K - \mathbf{M}\|_{\max} \leq C^{K+1}$, with $\mathbf{M}$ in Eq.(3.24).

[18] Due to space limitations, we only show the i-th and j-th columns of $\mathbf{S}$ here, which is sufficient for computing the new $\tilde{\mathbf{S}}$ in $G \cup \{(i, j)\}$.

Then, in light of $\boldsymbol{\gamma}$, Inc-uSR iteratively computes $\mathbf{M}_k$ via lines 13–17. After $K = 10$ iterations, $\mathbf{M}_K$ is derived as

	(a)	(b)	(c)	(d)	(e)	(f)	$\cdots$	(i)	(j)	$(k)\cdots(o)$
(a)	-0.005	-0.009	0	0.009					-0.009	
(b)	-0.004	-0.006	0	0.006			0		-0.007	0
(c)	0	0	0	0					0	
(d)	-0.002	-0.002	0	-0.005					0	
$\vdots$ (i)		0					0		0	0
(j)	0.028	0.037	0	0		-0.068		-0.104	-0.060	
$\vdots$ (o)		0					0		0	0

Finally, using $\mathbf{M}_K$ and the old $\mathbf{S}$, the new $\tilde{\mathbf{S}}$ is obtained via line 18, as partly shown in Column '$\mathsf{sim}_{\mathsf{true}}$' of Figure 3.1. □

Correctness & Complexity. (i) Algorithm Inc-uSR *correctly* updates the SimRank scores, which can be readily verified by Theorems 3.4–3.7. (ii) The total time of Inc-uSR can be bounded by $O(Kn^2)$ for updating *all* similarities of n^2 node-pairs.[19] To be specific, Inc-uSR runs in two phases: preprocessing (lines 1–12), and incremental iterations (lines 13–19). (a) For the preprocessing, it requires $O(m)$ time in total (m is the number of edges in the old G), which is dominated by computing $\mathbf{w}$ (lines 3), involving the matrix-vector multiplication $\mathbf{Q} \cdot [\mathbf{S}]_{\star,i}$. The time for computing vectors $\mathbf{u}, \mathbf{v}, \boldsymbol{\gamma}$ is bounded by $O(n)$, which only includes vector scaling and additions, *i.e.,* SAXPY. (b) For the incremental iterative phase, computing $\boldsymbol{\xi}_{k+1}$ and $\boldsymbol{\eta}_{k+1}$ needs $O(m+n)$ time for each iteration (lines 15–16). Computing $\mathbf{M}_{k+1}$ entails $O(n^2)$ time for performing one outer product of two vectors and one matrix addition (lines 17). Thus, the cost of this phase is $O(Kn^2)$ time for K iterations. Collecting (a) and (b), all n^2 node-pair similarities can

[19] In the next subsection, we shall further reduce the time complexity via a pruning strategy to eliminate node-pairs with unchanged similarities in $\boldsymbol{\Delta}\mathbf{S}$.

be incrementally computed in $O(Kn^2)$ total time, as opposed to the $O(r^4n^2)$ time of its counterpart [LHH+10] via SVD.

3.3.2 Pruning Unnecessary Node-Pairs in ΔS

After the SimRank update matrix $\mathbf{\Delta S}$ has been characterized in terms of a rank-one Sylvester equation, the pruning techniques in this subsection can further skip the node-pairs with unchanged similarities in $\mathbf{\Delta S}$ (*i.e.,* "unaffected areas"), avoiding unnecessary score recomputations for link update.

In practice, we observe that when link updates are small, affected areas in similarity updates $\mathbf{\Delta S}$ are often small as well. As demonstrated in Example 3.8, many entries in matrix $\mathbf{M}_K$ are 0s, implying that $\mathbf{\Delta S}$ $(= \mathbf{M}_K + \mathbf{M}_K^T)$ is a sparse matrix. However, it is a big challenge to identify such "affected areas" in $\mathbf{\Delta S}$ in response to link updates. To address this problem, we first introduce a nice property of the adjacency matrix:

Lemma 3.9. Let $\mathbf{A}$ be the adjacency matrix. The entry $[\mathbf{A}^k]_{i,j}$ counts the number of length-k paths from node i to j. □

For example, $[\mathbf{A}^4]_{i,j}$ counts the number of specific paths $\rho : i \to \circ \to \circ \to \circ \to j$ in G, with $\circ$ denoting any node.

Lemma 3.9 can be extended to count the number of "specific paths" whose edges are not necessarily in the same direction. For example, we can use $[\mathbf{A}\mathbf{A}^T\mathbf{A}\mathbf{A}^T]_{i,j}$ to count the paths $\rho : i \to \circ \leftarrow \circ \to \circ \leftarrow j$ in G, where $\mathbf{A}$ (*resp.*$\mathbf{A}^T$) appears at the positions 1,3 (*resp.*2,4), corresponding to the positions of $\to$ (*resp.*$\leftarrow$) in ρ.

As $\mathbf{Q}$ is the weighted (*i.e.,* row-normalized) matrix of $\mathbf{A}^T$, we can verify $[\mathbf{Q}^k \cdot (\mathbf{Q}^T)^k]_{i,j} = 0 \Leftrightarrow [(\mathbf{A}^T)^k \cdot \mathbf{A}^k]_{i,j} = 0$. The following corollary is immediate.

Corollary 3.10. Given $k = 0, 1, \cdots$, the entry $[\mathbf{Q}^k \cdot (\mathbf{Q}^T)^k]_{i,j}$ counts the weights of the specific paths whose left k edges in "$\leftarrow$" direction and right k edges in "$\to$" direction as follows:

$$\underbrace{i \leftarrow \circ \leftarrow \cdots \leftarrow}_{\text{length } k} \bullet \underbrace{\rightarrow \cdots \rightarrow \circ \rightarrow j}_{\text{length } k}. \quad \square \tag{3.31}$$

Definition 3.11. We call the paths in Eq.(3.31) *the symmetric in-link paths of length* $2k$ *for node-pair* (i, j). □

By virtue of Eq.(3.23), the recursive form of SimRank Eq.(2.2) naturally leads itself to the following series form:

$$[\mathbf{S}]_{a,b} = (1 - C) \cdot \sum_{k=0}^{\infty} C^k \cdot [\mathbf{Q}^k \cdot (\mathbf{Q}^T)^k]_{a,b}. \tag{3.32}$$

Capitalizing on Corollary 3.10, Eq.(3.32) provides a reinterpretation of SimRank: $[\mathbf{S}]_{a,b}$ is the weighted sum of all in-link paths of length $2k$ ($k = 0, 1, 2, \cdots$) for node-pair (a, b). The weight C^k in Eq.(3.32) is to reduce the contributions of in-link paths with *long* lengths relative to those with *short* ones. The factor $(1 - C)$ aims at normalizing $[\mathbf{S}]_{a,b}$ into $[0, 1]$ since $\| \sum_{k=0}^{\infty} C^k \cdot \mathbf{Q}^k \cdot (\mathbf{Q}^T)^k \|_{\max} \leq \sum_{k=0}^{\infty} C^k \leq \frac{1}{1-C}$.

Affected Areas in $\Delta\mathbf{S}$. In light of our interpretation for $\mathbf{S}$ via Eq.(3.32), we next reinterpret the series $\mathbf{M}$ in Theorem 3.7, with the aim to identify the "affected areas" in $\Delta\mathbf{S}$.

Due to space limitations, we shall mainly focus on the edge insertion case of $d_j > 0$. Other cases have the similar results.

By substituting Eq.(3.25) (the case $d_j > 0$) back into Eq.(3.24), we can readily split the series form of $\mathbf{M}$ into three parts:

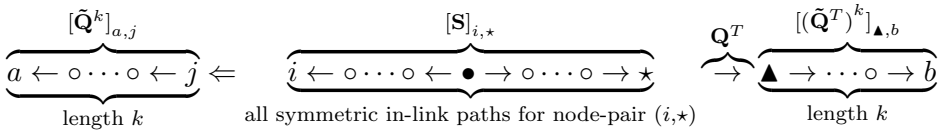
$$[\mathbf{M}]_{a,b} = \frac{1}{d_j + 1} \Bigg(\underbrace{\sum_{k=0}^{\infty} C^{k+1} \cdot [\tilde{\mathbf{Q}}^k]_{a,j} [\mathbf{S}]_{i,\star} \mathbf{Q}^T \cdot [(\tilde{\mathbf{Q}}^T)^k]_{\star,b}}_{\text{Part 1}} -$$

$$\underbrace{- \sum_{k=0}^{\infty} C^k [\tilde{\mathbf{Q}}^k]_{a,j} [\mathbf{S}]_{j,\star} [(\tilde{\mathbf{Q}}^T)^k]_{\star,b}}_{\text{Part 2}} + \mu \underbrace{\sum_{k=0}^{\infty} C^{k+1} [\tilde{\mathbf{Q}}^k]_{a,j} [(\tilde{\mathbf{Q}}^T)^k]_{j,b}}_{\text{Part 3}} \Bigg)$$

with the scalar $\mu := \frac{\lambda}{2(d_j+1)} + \frac{1}{C} - 1$.

By Lemma 3.9 and Corollary 3.10, when edge (i, j) is inserted and $d_j > 0$, Part 1 of $[\mathbf{M}]_{a,b}$ tallies the weighted sum of the following new paths for node-pair (a, b) in graph

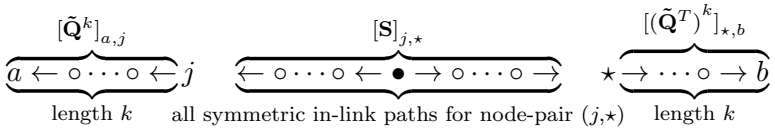
$G \cup \{(i,j)\}$:



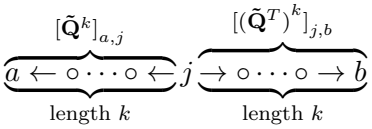
(3.33)

Such paths are the concatenation of four types of sub-paths (as depicted above) associated with four matrices, respectively, $[\tilde{\mathbf{Q}}^k]_{a,j}$, $[\mathbf{S}]_{i,\star}$, $\mathbf{Q}^T$, $[(\tilde{\mathbf{Q}}^T)^k]_{\blacktriangle,b}$, plus the inserted edge $j \Leftarrow i$. When such entire concatenated paths exist in the new graph, they should be accommodated for assessing the new SimRank $[\tilde{\mathbf{S}}]_{a,b}$ in response to the edge insertion (i,j) because our reinterpretation of SimRank indicates that SimRank counts *all* the symmetric in-link paths, and the entire concatenated paths can prove to be symmetric in-link paths.

Likewise, Parts 2 and 3 of $[\mathbf{M}]_{a,b}$, respectively, tally the weighted sum of the following new paths for node-pair (a,b):



(3.34)



(3.35)

Indeed, when edge (i,j) is inserted, only these three kinds of paths have extra contributions for $\mathbf{M}$ (therefore for $\mathbf{\Delta S}$). As incremental updates in SimRank merely tally these paths, node-pairs without having such paths could be safely pruned. In other words, for those pruned node-pairs, the three kinds of paths will have "zero contributions" to the changes in $\mathbf{M}$ in response to edge insertion. Thus, after pruning, the remaining node-pairs in G constitute the "affected areas" of $\mathbf{M}$.

To find the "affected areas" of $\mathbf{M}$, we prune the redundant node-pairs in G, based on the following theorem.

Theorem 3.12. *For the edge (i,j) insertion, let $\mathcal{O}(a)$ and $\tilde{\mathcal{O}}(a)$ be the out-neighbors of node a in old G and new $G \cup \{(i,j)\}$, respectively. Let $\mathbf{M}_k$ be the k-th iterative matrix*

in Line 17 of Algorithm 3.1, and let

$$\mathcal{F}_1 := \{b \mid b \in \mathcal{O}(y),\ \exists y,\ s.t.\ [\mathbf{S}]_{i,y} \neq 0\} \tag{3.36}$$

$$\mathcal{F}_2 := \begin{cases} \varnothing & (d_j = 0) \\ \{y \mid [\mathbf{S}]_{j,y} \neq 0\} & (d_j > 0) \end{cases} \tag{3.37}$$

$$\mathcal{A}_k \times \mathcal{B}_k := \begin{cases} \{j\} \times (\mathcal{F}_1 \cup \mathcal{F}_2 \cup \{j\}) & (k = 0) \\ \{(a,b) |\, a \in \tilde{\mathcal{O}}(x),\ b \in \tilde{\mathcal{O}}(y),\ \exists x,\ \exists y,\ s.t.\ [\mathbf{M}_{k-1}]_{x,y} \neq 0\} & (k > 0) \end{cases} \tag{3.38}$$

Then, for every iteration $k = 0, 1, \cdots$, the matrix $\mathbf{M}_k$ has the following sparse property:

$$[\mathbf{M}_k]_{a,b} = 0 \quad \textit{for all } (a,b) \notin (\mathcal{A}_k \times \mathcal{B}_k) \cup (\mathcal{A}_0 \times \mathcal{B}_0).$$

For the edge (i,j) deletion case, all the above results hold except that, in Eq.(3.37), the conditions $d_j = 0$ and $d_j > 0$ are, respectively, replaced by $d_j = 1$ and $d_j > 1$. □

Proof. We only show the edge insertion case for $d_j > 0$, due to space limitations. The proofs of other cases are similar.

For $k = 0$, it follows from Eq.(3.24) that $[\mathbf{M}_0]_{a,b} = [\mathbf{e}_j]_a [\boldsymbol{\gamma}]_b$. Thus, $\forall (a,b) \notin \mathcal{A}_0 \times \mathcal{B}_0$, there are two cases: (i) $a \neq j$, or (ii) $a = j$, $b \in {\mathcal{F}_1}^C \cap {\mathcal{F}_2}^C$, and $b \neq j$.

For case (i), $[\mathbf{e}_j]_a = 0$ since $a \neq j$. Thus, $[\mathbf{M}_0]_{a,b} = 0$. For case (ii), $[\mathbf{e}_j]_a = 1$ since $a = j$. Thus, $[\mathbf{M}_0]_{a,b} = [\boldsymbol{\gamma}]_b$, where $[\boldsymbol{\gamma}]_b$ is the linear combinations of the 3 terms: $[\mathbf{Q}]_{b,\star} \cdot [\mathbf{S}]_{\star,i}$, $[\mathbf{S}]_{b,j}$, and $[\mathbf{e}_j]_b$, according to the case of $d_j > 0$ in Eq.(3.25).

In the sequel, our goal is to show the 3 terms are all 0s. (a) For $b \notin \mathcal{F}_1$, by definition in Eq.(3.36), $b \in \mathcal{O}(y)$ for $\forall y$, we have $[\mathbf{S}]_{i,y} = 0$. Due to symmetry, $b \in \mathcal{O}(y) \Leftrightarrow y \in \mathcal{I}(b)$ [20], which implies that $[\mathbf{S}]_{i,y} = 0$ for $\forall y \in \mathcal{I}(b)$. Thus, $[\mathbf{Q}]_{b,\star} \cdot [\mathbf{S}]_{\star,i} = \frac{1}{\mathcal{I}(b)} \sum_{x \in \mathcal{I}(b)} [\mathbf{S}]_{x,i} = 0$. (b) For $b \notin \mathcal{F}_2$, it follows from the case $d_j > 0$ in Eq.(3.37) that $[\mathbf{S}]_{j,b} = 0$. Hence, by $\mathbf{S}$ symmetry, $[\mathbf{S}]_{b,j} = [\mathbf{S}]_{j,b} = 0$. (c) $[\mathbf{e}_j]_b = 0$ since $b \neq j$.

Taking (a)–(c) together, it follows that $[\mathbf{M}_0]_{a,b} = 0$, which completes the proof for the case $k = 0$.

[20] Recall that, as mentioned before, $\mathcal{I}(a)$ is the in-neighbor set of node a.

For $k > 0$, one can readily prove that the k-th iterative $\mathbf{M}_k$ in Line 17 of Algorithm 3.1 is the first k-th partial sum of $\mathbf{M}$ in Eq.(3.24). Thus, $\mathbf{M}_{k+1}$ can be derived from $\mathbf{M}_k$ as follows:

$$\mathbf{M}_k = C \cdot \tilde{\mathbf{Q}} \cdot \mathbf{M}_{k-1} \cdot \tilde{\mathbf{Q}}^T + C \cdot \mathbf{e}_j \cdot \boldsymbol{\gamma}^T.$$

Thus, the (a, b)-component form of the above equation is

$$[\mathbf{M}_k]_{a,b} = \frac{C}{|\tilde{\mathcal{I}}(a)||\tilde{\mathcal{I}}(b)|} \sum_{x \in \tilde{\mathcal{I}}(a)} \sum_{y \in \tilde{\mathcal{I}}(b)} [\mathbf{M}_{k-1}]_{x,y} + C \cdot [\mathbf{e}_j]_a \cdot [\boldsymbol{\gamma}]_b.$$

To show that $[\mathbf{M}_k]_{a,b} = 0$ for $(a, b) \notin \mathcal{A}_0 \times \mathcal{B}_0 \cup \mathcal{A}_k \times \mathcal{B}_k$, we follow the 2 steps: (i) For $(a, b) \notin \mathcal{A}_0 \times \mathcal{B}_0$, as proved in the case $k = 0$, the term $C \cdot [\mathbf{e}_j]_a [\boldsymbol{\gamma}]_b$ in the above equation is obviously 0. (ii) For $(a, b) \notin \mathcal{A}_k \times \mathcal{B}_k$, by virtue of Eq.(3.38), $a \in \tilde{\mathcal{O}}(x), b \in \tilde{\mathcal{O}}(y)$, for $\forall x, y$, we have $[\mathbf{M}_{k-1}]_{x,y} = 0$. Hence, by symmetry, it follows that $x \in \tilde{\mathcal{I}}(a), y \in \tilde{\mathcal{I}}(b)$, $[\mathbf{M}_{k-1}]_{x,y} = 0$.

Taking (i) and (ii) together, we conclude that

$$[\mathbf{M}_k]_{a,b} = 0 \text{ for } (a, b) \notin \mathcal{A}_0 \times \mathcal{B}_0 \cup \mathcal{A}_k \times \mathcal{B}_k,$$

which completes the proof. □

Theorem 3.12 provides a pruning strategy to iteratively eliminate node-pairs with a-priori zero values in $\mathbf{M}_k$ (thus in $\mathbf{\Delta S}$). Hence, by leveraging Theorem 3.12, when edge (i, j) is updated, we just need to consider node-pairs in $(\mathcal{A}_k \times \mathcal{B}_k) \cup (\mathcal{A}_0 \times \mathcal{B}_0)$ for incrementally updating $\mathbf{\Delta S}$.

Intuitively, $\mathcal{F}_1$ in Eq.(3.36) captures the nodes "▲" in (3.33). To be specific, $\mathcal{F}_1$ can be obtained via 2 phases: (i) For the given node i, we first build an intermediate set $\mathcal{T} := \{y | [\mathbf{S}]_{i,y} \neq 0\}$, which consists of nodes "⋆" in (3.33). (ii) For each node $x \in \mathcal{T}$, we then find all out-neighbors of x in G, which produces $\mathcal{F}_1$, *i.e.,* , $\mathcal{F}_1 = \bigcup_{x \in \mathcal{T}} \mathcal{O}(x)$. Analogously, the set $\mathcal{F}_2$ in Eq.(3.37), in the case of $d_j > 0$, consists of the nodes "⋆" depicted in (3.34). When $d_j = 0$, $\mathcal{F}_2 = \varnothing$ since the term $[\mathbf{S}]_{\star,i}$ does not appear in the expression of $\boldsymbol{\gamma}$ in Eq.(3.25) for the case when $d_j = 0$, in contrast with the case $d_j > 0$.

After obtaining $\mathcal{F}_1$ and $\mathcal{F}_2$, we can readily find $\mathcal{A}_0 \times \mathcal{B}_0$, according to Eq.(3.38). For $k > 0$, to iteratively derive the node-pair set $\mathcal{A}_k \times \mathcal{B}_k$, we take the following two steps: (i) we first construct a node-pair set $\mathcal{T}_1 \times \mathcal{T}_2 := \{(x, y) | [\mathbf{M}_{k-1}]_{x,y} \neq 0\}$. (ii) For every node $x \in \mathcal{T}_1$ (*resp.*$y \in \mathcal{T}_2$), we then find all out-neighbors of x (*resp.*y) in $G \cup \{(i, j)\}$, which yields $\mathcal{A}_k$ (*resp.*$\mathcal{B}_k$), *i.e.*, $\mathcal{A}_k = \bigcup_{x \in \mathcal{T}_1} \tilde{\mathcal{O}}(x)$ and $\mathcal{B}_k = \bigcup_{y \in \mathcal{T}_2} \tilde{\mathcal{O}}(y)$.

The node selectivity of Theorem 3.12 hinges on $\mathbf{\Delta S}$ sparsity. Since real graphs are constantly updated with *minor* changes, $\mathbf{\Delta S}$ is often *sparse* in general. Hence, a huge body of node-pairs with zero scores in $\mathbf{\Delta S}$ can be eliminated in practice. As demonstrated by our experiments in Figure 3.5, 76.3% paper-pairs on DBLP can be pruned, significantly reducing unnecessary similarity recomputations in response to link updates.

Example 3.13. Recall Example 3.8 and the old graph G in Figure 3.1. When edge (i, j) is inserted to G, according to Theorem 3.12, $\mathcal{F}_1 = \{a, b\}$, $\mathcal{F}_2 = \{f, i, j\}$, $\mathcal{A}_0 \times \mathcal{B}_0 = \{j\} \times \{a, b, f, i, j\}$. Hence, instead of computing the entire vector $\boldsymbol{\gamma}$ in Eq.(3.25), we only need to compute part of its entries $[\boldsymbol{\gamma}]_x$ for $\forall x \in \mathcal{B}_0$.

For the first iteration, since $\mathcal{A}_1 \times \mathcal{B}_1 = \{a, b\} \times \{a, b, d, j\}$, then we only need to compute 18 ($= 3 \times 6$) entries $[\mathbf{M}_1]_{x,y}$ for $\forall (x, y) \in \{a, b, j\} \times \{a, b, d, f, i, j\}$, skipping the computations of 207 ($= 15^2 - 18$) remaining entries in $\mathbf{M}_1$. After $K = 10$ iterations, many unnecessary node-pairs are pruned, as in part highlighted in the gray rows of the table in Figure 3.1. □

Algorithm. We provide a complete incremental algorithm for computing SimRank, referred to as Inc-SR (in Algorithm 3.2), by incorporating our pruning strategy into Inc-uSR.

Correctness. The algorithm Inc-SR can *correctly* prune the node-pairs with a-priori zero scores in $\mathbf{\Delta S}$, which is verified by Theorem 3.12. It also *correctly* returns the new similarities, as evidenced by Theorems 3.4–3.7.

Complexity. The total time of Inc-SR is $O(K(nd + |\mathsf{AFF}|))$ for K iterations, where d is

Algorithm 3.2: Inc-SR $(G, \mathbf{S}, K, (i,j), C)$

Input / Output: the same as Algorithm 3.1.

1-2 the same as Algorithm 3.1 ;

3 find $\mathcal{B}_0$ via Eq.(3.38) ;

memoize $[\mathbf{w}]_b := [\mathbf{Q}]_{b,\star} \cdot [\mathbf{S}]_{\star,i}$, for all $b \in \mathcal{B}_0$;

4-12 almost the same as Algorithm 3.1 except that the computations of the entire vector $\boldsymbol{\gamma}$ in Lines $6, 8, 10, 12$ are replaced by the computations of only parts of entries in $\boldsymbol{\gamma}$, respectively, *e.g.,* in Line 6 of Algorithm 3.1, "$\boldsymbol{\gamma} := \mathbf{w} + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot \mathbf{e}_j$" are replaced by "$[\boldsymbol{\gamma}]_b := [\mathbf{w}]_b + \frac{1}{2}[\mathbf{S}]_{i,i} \cdot [\mathbf{e}_j]_b$, for all $b \in \mathcal{B}_0$" ;

13 set $[\boldsymbol{\xi}_0]_j := C$, $[\boldsymbol{\eta}_0]_b := [\boldsymbol{\gamma}]_b$, $[\mathbf{M}_0]_{j,b} := C \cdot [\boldsymbol{\gamma}]_b, \forall b \in \mathcal{B}_0$;

14 **for** $k = 1, \cdots, K$ **do**

15 find $\mathcal{A}_k \times \mathcal{B}_k$ via Eq.(3.38) ;

16 memoize $\sigma_1 := C \cdot (\mathbf{v}^T \cdot \boldsymbol{\xi}_{k-1})$, $\sigma_2 := \mathbf{v}^T \cdot \boldsymbol{\eta}_{k-1}$;

17 $[\boldsymbol{\xi}_k]_a := C \cdot [\mathbf{Q}]_{a,\star} \cdot \boldsymbol{\xi}_{k-1} + \sigma_1 \cdot [\mathbf{u}]_a$, for all $a \in \mathcal{A}_k$;

18 $[\boldsymbol{\eta}_k]_b := [\mathbf{Q}]_{b,\star} \cdot \boldsymbol{\eta}_{k-1} + \sigma_2 \cdot [\mathbf{u}]_b$, for all $b \in \mathcal{B}_k$;

19 $[\mathbf{M}_k]_{a,b} := [\boldsymbol{\xi}_k]_a \cdot [\boldsymbol{\eta}_k]_b + [\mathbf{M}_{k-1}]_{a,b}$, $\forall (a,b) \in \mathcal{A}_k \times \mathcal{B}_k$;

20 $[\tilde{\mathbf{S}}]_{a,b} := [\mathbf{S}]_{a,b} + [\mathbf{M}_K]_{a,b} + [\mathbf{M}_K]_{b,a}$, $\forall (a,b) \in \mathcal{A}_K \times \mathcal{B}_K$;

21 **return** $\tilde{\mathbf{S}}$;

the average in-degree of G, and $|\mathsf{AFF}| := \text{avg}_{k\in[0,K]}(|\mathcal{A}_k| \cdot |\mathcal{B}_k|)$ with $\mathcal{A}_k, \mathcal{B}_k$ in Eq.(3.38), being the average size of "affected areas" in $\mathbf{M}_k$ for K iterations. More concretely, (a) for the preprocessing, finding $\mathcal{B}_0$ (line 3) needs $O(dn)$ time. Utilizing $\mathcal{B}_0$, computing $[\mathbf{w}]_b$ reduces from $O(m)$ to $O(d|\mathcal{B}_0|)$ time, with $|\mathcal{B}_0| \ll n$. Analogously, $\boldsymbol{\gamma}$ in lines 6,8,10,12 of Algorithm 3.1 needs only $O(|\mathcal{B}_0|)$ time. (b) For each iteration, finding $\mathcal{A}_k \times \mathcal{B}_k$ (line 15) entails $O(dn)$ time. Memoizing σ_1, σ_2 needs $O(n)$ time (line 16). Computing $\boldsymbol{\xi}$ (*resp.*$\boldsymbol{\eta}$) reduces from $O(m)$ to $O(d|\mathcal{A}_k|)$ (*resp.*$O(d|\mathcal{B}_k|)$) time (lines 17–18). Computing $[\mathbf{M}_k]_{a,b}$ reduces from $O(n^2)$ to $O(|\mathcal{A}_k||\mathcal{B}_k|)$ time (line 19). Thus, the total time complexity can be bounded by $O(K(nd + |\mathsf{AFF}|))$ for K iterations.

It is worth mentioning that Inc-SR, in the worst case, has the same complexity bound of Inc-uSR. However, in practice, $|\mathsf{AFF}| \ll n^2$, as demonstrated by our experimental study in Figure 3.6, since real graphs are constantly updated with *small* changes. Hence, $O(K(nd + |\mathsf{AFF}|))$ is generally much smaller than $O(Kn^2)$. In the next section, we shall further confirm the efficiency of Inc-SR by conducting extensive experiments.

3.4 Experimental Evaluation

We present an empirical study, using real and synthetic data, to show (i) the efficiency of Inc-SR for incremental computation in terms of time and space, as compared with (a) Inc-SVD, the best known link-update algorithm [LHH+10], (b) Inc-uSR, our incremental algorithm without pruning, and (c) Batch, the batch algorithm [YLZ+13b] via fine-grained memoization; (ii) the effectiveness of our pruning technique for identifying "affected areas" to speed up Inc-SR computation; and (iii) the exactness of Inc-SR and Inc-uSR, in contrast with Inc-SVD.

3.4.1 Experimental Setting

Datasets. We use both real and synthetic datasets.

(1) DBLP[21], a co-citation graph, where each node is a paper with attributes (*e.g.,* publication year), and edges are citations. By virtue of the year of the papers, we extract dense snapshots, each consisting of 93,560 edges and 13,634 nodes.

(2) CitH[22], a reference network (cit-HepPh) from e-Arxiv. If a paper u references v, the graph has one link from u to v. The dataset has 421,578 edges and 34,546 nodes.

(3) YouTu[23], a YouTube graph, where each node is a video. A video u is linked to v if v is in the related video list of u. We extract snapshots according to the age of the videos, and each has 953,534 edges and 178,470 nodes.

[21] http://dblp.uni-trier.de/~ley/db/
[22] http://snap.stanford.edu/data/
[23] http://netsg.cs.sfu.ca/youtubedata/

We use GraphGen[24] to build synthetic graphs and updates. The graphs are controlled by (a) the number of nodes $|V|$, and (b) the number of edges $|E|$. We produce the sequence of graphs following the linkage generation model [GGCM09]. Two parameters are utilized to control the updates: (a) update type (edge insertion/deletion), and (b) the size of updates $|\Delta G|$.

All the algorithms are implemented in Visual C++ v10.0. Each experiment is run 5 times; we report the average here. We use a machine with an Intel Core(TM) 2.80 GHz CPU and 8GB RAM, running Windows 7.

We set the decay factor $C = 0.6$, as in the prior work [JW02]. Our default iteration number is set to $K = 15$, with which a high accuracy $C^K \leq 0.0005$ is attainable, according to [LVGT08]; on large dataset YouTu, K is set to 5, the same value as [JW02]. For Inc-SVD, the target rank r is a time-accuracy trade-off; as shown in the experiments [LHH+10], the highest speedup is achieved when $r = 5$. Thus, in our time evaluations, $r = 5$ is adopted, whereas in the exactness evaluations, we shall tune this value.

3.4.2 Experimental Results

Exp-1: Time Efficiency.

We first evaluate the running time of Inc-SR, Inc-uSR against Inc-SVD and Batch on real data.

To favor Inc-SVD that only works on graphs of small sizes (due to memory crash for high-dimension SVD, *e.g.,* $n > 10^5$), DBLP and CitH are used, though Inc-SR works well on a variety of graphs (*e.g.,* YouTu, Syn).

Figure 3.2 depicts the results for edges inserted into DBLP, CitH, YouTu, respectively. For each dataset, we fix $|V|$, and increase $|E|$ by $|\Delta E|$, as shown in the x-axis. Here, the edge updates are the differences between snapshots *w.r.t.* the "year" (*resp.*"video age") attribute of DBLP, CitH (*resp.*YouTu), reflecting their real-world evolution. We observe the following. (1) Inc-SR *always* outperforms Inc-SVD and Inc-uSR when edges

[24] http://www.cse.ust.hk/graphgen/

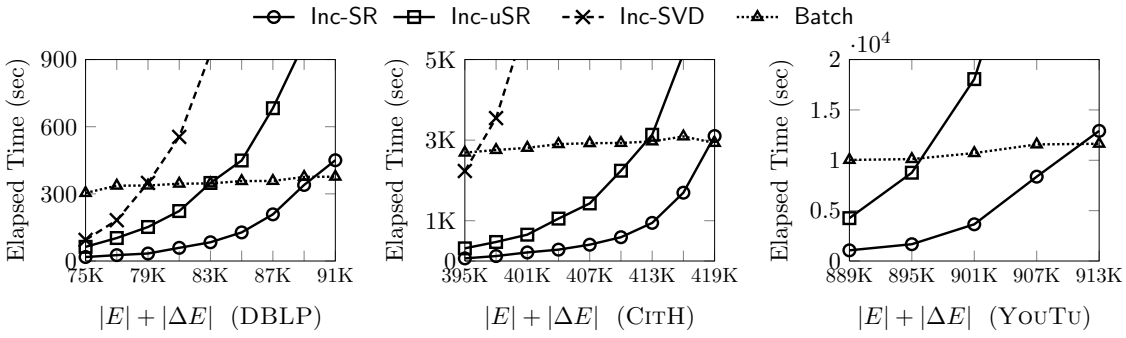


Figure 3.2: Time Efficiency of Incremental SimRank on Real Data

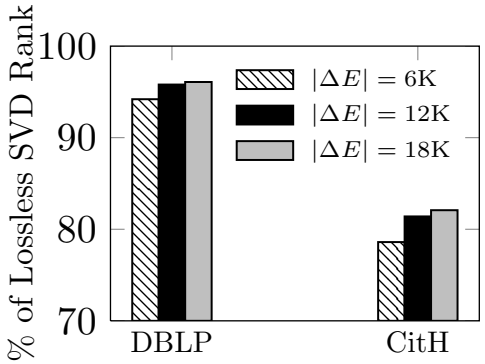


Figure 3.3: % of Lossless SVD Rank *w.r.t.* $|\Delta E|$

are increased. For example, on DBLP, when the edge changes are 10.7%, the time for Inc-SR (83.7s) is 11.2x faster than Inc-SVD (937.4s), and 4.2x faster than Inc-uSR (348.7s). This is because Inc-SR deploys a rank-one matrix trick to update the similarities, with an effective pruning strategy to skip unnecessary recomputations, as opposed to Inc-SVD that entails rather expensive costs to incrementally update the SVD. The results on CitH are more pronounced, *e.g.,* Inc-SR is about 30x better than Inc-SVD when $|E|$ is increased to 401K. On YouTu, Inc-SVD fails due to the memory crash for SVD. (2) Inc-SR is consistently better than Batch when the edge changes are fewer than 19.7% on DBLP, and 7.2% on CitH. When the link updates are 5.3% on DBLP (*resp.*3.9% on CitH), Inc-SR improves Batch by 10.2x (*resp.*4.9x). This is because (i) Inc-SR exploits the sparse structure of $\mathbf{\Delta S}$ for incremental update, and (ii) small link perturbations may keep $\mathbf{\Delta S}$ sparsity. Hence, Inc-SR is highly efficient when link updates are small. (3) The running time of Inc-SR, Inc-uSR, Inc-SVD, unlike Batch, is sensitive to the edge updates $|\Delta E|$,

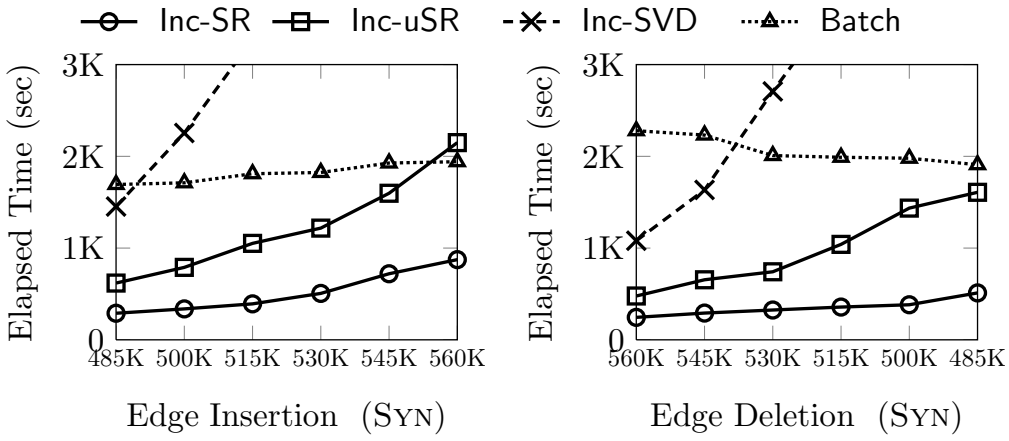


Figure 3.4: Time Efficiency of Incremental SimRank on Synthetic Data

as expected. The reason is that Batch needs to reassess all similarities from scratch in response to link updates, whereas Inc-SR and Inc-uSR can reuse the old information in SimRank for incremental updates. In addition, Inc-SVD is too sensitive to $|\Delta E|$, as it needs costly tensor products to compute SimRank from the updated SVD matrices. In contrast, Inc-SR is less sensitive than Inc-SVD as it *directly* computes SimRank changes *w.r.t.* link updates, without the need of computing SVD.

Figure 3.3 shows the target rank r required for the *lossless* SVD of Eq.(3.3) *w.r.t.* the edge changes $|\Delta E|$ on DBLP and CitH. The y-axis is $\frac{r}{n} \times 100\%$, where $n = |V|$, and r is the rank of the lossless SVD for $\mathbf{C}$ in Eq.(3.3). On each dataset, when increasing $|\Delta E|$ from 6K to 18K, we see that $\frac{r}{n}$ is 95% on DBLP (*resp.*80% on CitH), Thus, r is not negligibly smaller than n in real graphs. Due to the quartic time *w.r.t.* r, Inc-SVD may be slow in practice to get a high accuracy.

Fixing $|V| = 79,483$ on synthetic data, we vary $|E|$ from 485K to 560K (*resp.*560K to 485K) edges in 15K increments (*resp.*decrements). The results are reported in Figure 3.4, confirming our observations on real datasets. For example, when 6.4% edges are increased, Inc-SR runs 8.4x faster than Inc-SVD, 4.7x faster than Batch, and 2.7x faster than Inc-uSR. When 8.8% edges are deleted, Inc-SR outperforms Inc-SVD by 10.4x, Batch by 5.5x, and Inc-uSR by 2.9x. This justifies the complexity analysis of our algorithms Inc-SR and Inc-uSR.

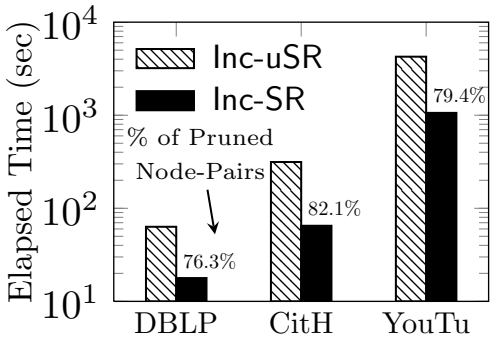


Figure 3.5: Effect of Pruning

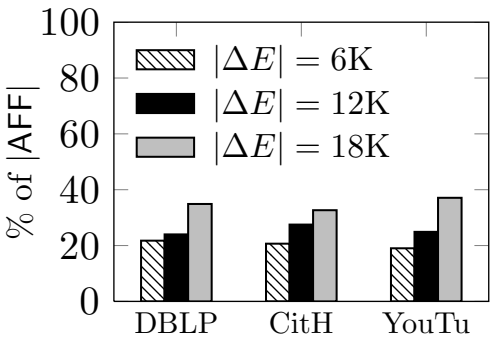


Figure 3.6: % of Affected Areas *w.r.t.* $|\Delta E|$

Exp-2: Effects of Pruning.

As mentioned in Subsection 3.3.2, Inc-SR skips needless computations for incremental updates. To show the effectiveness of our pruning strategy in Inc-SR, we compare its time with that of Inc-uSR, *i.e.,* original version of Inc-SR without pruning rules, on DBLP, CitH, YouTu. The results are shown in Figure 3.5, where the percentage of the pruned node-pairs in each graph is depicted on the black bar. The y-axis is in a logarithmic scale. It can be discerned that, on every dataset, Inc-SR constantly outperforms Inc-uSR by nearly 0.5 order of magnitude. For instance, the running time of Inc-SR (64.9s) improves that of Inc-uSR (314.2s) by 4.8x on CitH, with approximately 82.1% node-pairs being pruned. That is, our pruning technique is effective in finding unnecessary node-pairs on real graphs with various link distributions.

As our pruning strategy hinges on the size of the "affected areas" in SimRank update matrix, it is imperative to evaluate, on real graphs, that how large these "affected areas" are when links are evolved. The results are visualized in Figure 3.6, showing that the percentage of the "affected areas" in similarity changes *w.r.t.* link updates $|\Delta E|$ on real DBLP, CitH, and YouTu. We find the following. (1) When $|\Delta E|$ is varied from 6K to 18K on every real dataset, the "affected areas" in similarity changes are relatively small. For instance, when $|\Delta E| = 12K$, the percentage of the "affected areas" is only 23.9% on DBLP, 27.5% on CitH, and 24.8% on YouTu, respectively. This demonstrates the potential benefits of our pruning technique in real applications, where a larger number

of elements in $\mathbf{\Delta S}$ with a-priori zero scores can be pruned. (2) For each dataset, the size of "affect areas" mildly grows when $|\Delta E|$ is increased. For example, on YouTu, the percentage of $|\mathsf{AFF}|$ increases from 19.0% to 24.8% when $|\Delta E|$ is changed from 6K to 12K. This confirms our observation in the time efficiency analysis, where Inc-SR speedup is more obvious for smaller $|\Delta E|$.

Exp-3: Memory Space.

We next evaluate the memory requirements of Inc-SR, Inc-uSR, against Inc-SVD on real datasets. Here, the memory space means "*intermediate* space", where the last step of writing n^2 node-pairs of the similarity outputs is not accommodated. We also tune the default target rank $r = 5$ larger for Inc-SVD to see how memory increases *w.r.t.* r.

The results are depicted in Figure 3.7, where, for Inc-SVD, we report $r = 15, 25$ on only small DBLP, as its memory space will explode on larger networks when r and $|V|$ grow. We notice that (1) Inc-SR and Inc-uSR consume far smaller space than Inc-SVD by at least 1.5 orders of magnitude on DBLP and CitH no matter what target rank r might be. This is because Inc-SR and Inc-uSR use the rank-one trick to convert $\mathbf{\Delta S}$ computations into the sequence of *vector* operations, whereas Inc-SVD needs to memoize the decomposed SVD matrices and to perform costly matrix tensor products. (2) Inc-SR has 4.1x (*resp.*4.5x) smaller space than Inc-uSR on DBLP (*resp.*YouTu), due to our pruning method reducing the memoization of many entries in auxiliary vectors, *e.g.,* $\mathbf{w}$. (3) When r is varied from 5 to 25, the space of Inc-SVD is increased from 637.9M to 3.15G on DBLP, but crashes on CitH and YouTu. This tells that r has a large impact on the performance of Inc-SVD, which cannot be ignored in the big-O notation of the complexity analysis [LHH+10]. Thus, to get Inc-SVD feasible on CitH, we set $r = 5$ in the evaluations.

Exp-4: Exactness.

Finally, we evaluate the exactness of Inc-SR and Inc-uSR against Inc-SVD. We adopt the NDCG metrics [LHH+10] to assess top-30 most similar node-pairs on DBLP, CitH,

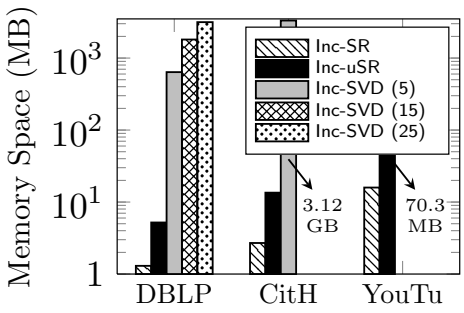


Figure 3.7: Memory Space

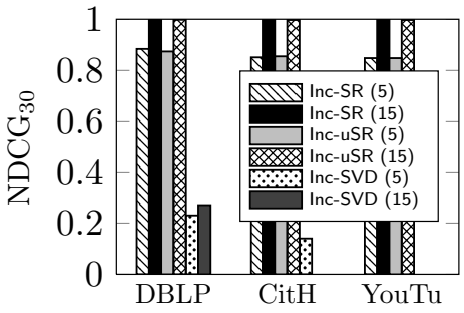


Figure 3.8: $NDCG_{30}$ Exactness

YouTu. For baselines of $NDCG_{30}$, we use the results of Batch on each dataset for 35 iterations.[25] For Inc-SR and Inc-uSR, we perform $K = 5, 15$ iterations on each graph; for Inc-SVD, due to its non-iterative paradigm, we tune the rank r from 5 to 15. The results are depicted in Figure 3.8, telling us the following. (1) In all the cases, Inc-SR and Inc-uSR have much better accuracy than Inc-SVD. For example, the $NDCG_{30}$ of Inc-SR and Inc-uSR are both 0.88 at $K = 5$, much better than Inc-SVD (0.36) at $r = 25$. This confirms our observations in Section 3.2, where we envision that Inc-SVD may miss some eigen-information in many real graphs. When $K = 10$, the $NDCG_{30}$ of Inc-SR and Inc-uSR are 1s, indicating that their top-30 node-pairs are perfectly accurate. This justifies the correctness of our algorithms. (2) For each dataset and the fixed iteration K, the $NDCG_{30}$ of Inc-SR and Inc-uSR are exactly the same. This indicates that our pruning strategy is lossless, *i.e.,* it does not sacrifice any exactness for speedup.

3.5 Related Work

SimRank is arguably one of the most appealing link-based similarity measures in a graph. Recent results on SimRank computation can be distinguished into two broad categories: (i) incremental SimRank on dynamic graphs (*e.g.,* [HFLC10, LHH+10]), and (ii) batch SimRank on static graphs (*e.g.,* [FNSO13, LLY+10, FR07, LLY12, LVGT08, YLZ+13b]).

[25] As the diameters (*i.e.,* the longest paths) of DBLP, CitH, YouTu are 16,11,7, respectively, it suffices to perform $K = 35$ iterations to accommodate *all* path-pairs between two nodes for assessing SimRank. Thus, the resulting scores of Batch for $K = 35$ can be viewed as the *exact* baseline solutions.

3.5.1 Incremental Update

Incremental computation is useful as real graphs are often updated with small changes. However, few results are known about incremental SimRank computation, much less than their batch counterparts (*e.g.,* [YMS14, AMC08, FR07, LLY12, LLY+10, LVGT08, YLZ+13b, FNSO13]). Generally, there are two types of updates for dynamic graphs: (i) link updates, and (ii) node updates. About link-incremental SimRank algorithms, we are merely aware of [LHH+10] by Li *et al.* who gave an excellent matrix representation of SimRank, and was the first to show a SVD method for incrementally updating similarities of all node-pairs in $O(r^4n^2)$ time [26], where r ($\leq n$) is the target rank of the low-rank approximation. However, (i) their incremental approach is *inherently* inexact, without guaranteed accuracy. It may miss some eigen-information (as proved in Section 3.2) even though r is chosen to be exactly the rank (instead of low-rank) of the target matrix for the lossless SVD. (ii) In practice, r is not *much* smaller than n for attaining the desirable accuracy. This may lead to prohibitively expensive updating costs for [LHH+10] since its time complexity $O(r^4n^2)$ is quartic *w.r.t.* r. In comparison, our work adopts a completely different framework from [LHH+10]. Instead of incrementally updating SVD [LHH+10], we characterize the changes to SimRank in response to each link update as a rank-one Sylvester equation first, and then exploit the link structure to prune "unaffected areas" for speeding up the incremental computation of SimRank, without loss of exactness, which only needs constant time (independent of r) to incrementally compute every node-pair similarity for each link update.

Another interesting piece of work is due to He *et al.* [HFLC10], who devised the parallel computation of SimRank on digraphs, by leveraging the iterative aggregation strategy. Indeed, the parallel computing technique in [HFLC10] can be regarded as an efficient *node*-incremental updating framework for SimRank. It differs from this work in that [HFLC10] improves the efficiency by reordering and parallelizing the first-order

[26] According to the *proof* of Lemma 2 in [LHH+10], the time is actually $O(r^4n^2)$, though, the statement of Lemma 2 says "it is bounded by $O(n^2)$". Observing that $r \ll n$ is not often the case, we do not explicitly omit r^4 in $O(\star)$ here.

Markov chain for node updates on GPU, instead of capturing the "unaffected areas" of SimRank *w.r.t.* link updates, whereas our methods utilize pruning rules to eliminate unnecessary recomputations for links updates on CPU via a rank-one Sylvestor equation.

There has also been work on the incremental computations for other hyperlink-based relevance measures (*e.g.,* [DPSK05, BCG10, SGP11, FNOK12, YLZ+12, YLZ13a]). Desikan *et al.* [DPSK05] proposed an efficient incremental PageRank algorithm for node updating. Their underlying principle is based on the first-order Markov model. Banhmani *et al.* [BCG10] utilized the Monte Carlo method for incrementally computing Personalized PageRank. Yu *et al.* [YLZ+12] provided an incremental eigenvector update algorithm for SimFusion+ computation. Sarma *et al.* [SGP11] gave an excellent exposition of concatenating the short random walks for estimating PageRank with provable accuracy on graph streams. All of these incremental methods are *probabilistic* in nature, with the focus on node ranking, and hence cannot be directly applied in SimRank node-pair scoring. Fujiwara *et al.* [FNOK12] proposed K-dash for finding top-k highest Random Walk with Restart (RWR) proximity nodes for a given query, which involves a strategy to incrementally *estimate* proximity bounds. However, their incremental process is *approximate*. Later, Yu *et al.* [YLZ13a] proposed an incremental strategy for RWR link updates.

3.5.2 Batch Computation

In contrast to the incremental algorithm, the batch SimRank computation has been well-studied on static graphs. Recent results on batch SimRank can be mainly categorized into (i) deterministic computation (*e.g.,* [JW02, FNSO13, LHH+10, LVGT08, YLZ+13b]), and (ii) probabilistic estimation (*e.g.,* [FR07, LLY12, LLY+10]). The deterministic methods may obtain similarities of high accuracy, but the time complexity is less desirable than the probabilistic approaches.

For deterministic methods, Jeh and Widom [JW02] are the first to propose an iterative paradigm for SimRank, entailing $O(Kd^2n^2)$ time for K iterations, where d is the average in-degree. Later, Lizorkin *et al.* [LVGT08] utilized the partial sums memoization to

speed up SimRank computation to $O(Kdn^2)$. Li *et al.* [LHH+10] proposed a novel non-iterative matrix formula for SimRank. Apart from the incremental SimRank requiring $O(r^4n^2)$ time, they also used a SVD method for computing batch SimRank in $O(\alpha^4n^2)$ time, where α is the target rank of matrix $\mathbf{Q}$. Most recently, Yu *et al.* [YLZ+13b] have further improved SimRank computation to $O(Kd'n^2)$ time (with $d' \leq d$) via a fine-grained memoization to share the common parts among different partial sums. Fujiwara *et al.* [FNSO13] exploited the matrix form of [LHH+10] to find the top-k similar nodes in $O(n)$ time.

For probabilistic approaches, Fogaras and Rácz [FR07] proposed P-SimRank in linear time to estimate $s(a, b)$ from the probability that two random surfers, starting from a and b, will finally meet at a node. Li *et al.* [LLY+10] harnessed the random walks to compute local SimRank for a single node-pair. Lee *et al.* [LLY12] deployed the Monte Carlo method to find top-k SimRank node-pairs.

3.6 Conclusions

In this chapter, we have proposed an efficient algorithm for incrementally computing SimRank on link-evolving graphs. Our algorithm, Inc-SR, is based on two key ideas: (1) The SimRank update matrix $\mathbf{\Delta S}$ is characterized via a rank-one Sylvester equation. Based on this, a novel efficient paradigm is devised, which improves the incremental computation of SimRank from $O(r^4n^2)$ to $O(Kn^2)$ for every link update. (2) An effective pruning strategy is proposed to skip unnecessary similarity recomputations for link updates, further reducing the computation time of SimRank to $O(K(nd + |\mathsf{AFF}|))$, where $|\mathsf{AFF}|$ ($\leq n^2$) is the size of "affected areas" in SimRank update matrix, which can be practically much smaller than n^2 in real evolution. Our empirical evaluations show that (1) Inc-SR consistently outperforms the best known link-update algorithm [LHH+10], from several times to over one order of magnitude, without loss of exactness. (2) Inc-SR runs substantially faster than its batch counterpart when link updates are small.

Chapter 4

Efficient Penetrating-Rank on Large Networks

4.1 Introduction

The problem of quantifying relevance between entities based has witnessed growing interests over the last decades. There are various circumstances in which it would be useful to answer the questions such as "How similar are every two entities (vertices)?", and "Which other entities (vertices) are most similar to a specific query (a given query vertex)?". Unlike SimRank [JW02] that considers in-link relationships alone for relevance assessment, Penetrating-Rank (P-Rank) has been recently proposed as another promising similarity measure, which was invented by Zhao *et al.* [ZHS09]. It encodes both incoming and outgoing links of entities into similarity assessment. P-Rank similarities flowing from in-link neighbors of entities are penetrated through their out-link neighbors in a recursive fashion. In contrast to other similarity measures, P-Rank has the following advantages:

- **Semantic Completeness.** P-Rank provides a comprehensive way of jointly considering both in- and out-link relationships with semantic completeness. In comparison, other similarity measures (*e.g.,* SimRank and SimFusion) have the "limited information problem" [JW02], in which only in-links are partially exploited, whereas

out-links are totally ignored.

- **Adaptivity.** P-Rank can be easily combined with other textual domain-specific similarity metrics to produce an overall similarity measure, which is fairly adaptive to any domains with entity-to-entity relationships.

- **Generality.** P-Rank formula has a general form that makes itself transcend other existing similarity measures. For instance, SimRank [JW02] and Amsler [Ams72] are just special cases of P-Rank, by setting specific weight factors for P-Rank, as illustrated in [ZHS09].

Therefore, P-Rank has long been recognized as an important and common similarity measure, which has a wide range of applications in any fields where other similarity measures (*e.g.,* SimRank, SimFusion, and Amsler) are applicable, such as collaborative filtering, graph clustering, link prediction, and web document ranking (*e.g.,* [LVGT10, ZCY09, AMC08] and references therein).

4.1.1 Motivation

Previous work on P-Rank, however, leaves several challenging issues unaddressed, motivating us to systematically develop efficient techniques for P-Rank assessment.

Firstly, it is not straightforward to estimate the accuracy for P-Rank iterations. Although the convergence of P-Rank iterations has been proved in [ZHS09], it is still difficult to determine the total number of iterations needed for guaranteeing a given accuracy. To the best of our knowledge, there is only one work [LVGT10] that estimates the accuracy for SimRank iterations, which is a special case of P-Rank (only in-links are considered). That work provides an upper bound C^{k+1} for the difference between the k-th iterative SimRank and the exact one. However, there is extra difficulty in porting this bound to P-Rank accuracy estimation since we observe that the simple linear combination of the weighted bound $\lambda \cdot {C_{\text{in}}}^{k+1} + (1-\lambda) \cdot {C_{\text{out}}}^{k+1}$ is not always suitable for P-Rank (unless in-link damping factor C_{in} equals out-link damping factor C_{out}). The

reason is that the recursive nature of P-Rank makes both in- and out-links have a *recursive* impact on similarities of all pairs of vertices. Thus, it is imperative to derive a new bound for estimating the accuracy of P-Rank.

Secondly, no prior work has studied the stability of P-Rank. In the iterative computation, analyzing P-Rank stability plays a paramount role because it not only can gauge the sensitivity of similarity results to slight perturbations in the link structure (*e.g.,* by adding or removing edges) but also implies whether large amounts of accumulated round-off errors in the P-Rank iterations will run the risk of producing nonsensical similarity results. To address this issue, we provide an upper bound for the P-Rank condition number that is defined over the closed-form of P-Rank. One complicated problem is to calculate the norm of a large matrix inversion in the closed-form of P-Rank since the straightforward computation is prohibitively expensive. Motivated by this, we propose a new approach to obtain a neat tight bound for the P-Rank condition number, which can avoid the computation of such a large matrix inversion.

Thirdly, it is a big challenge to improve the computational complexity of P-Rank. The naive method computing P-Rank via the fixed-point iteration requires $O(Kn^4)$ time for K iterations, which is inapplicable to large networks. The most efficient existing technique using partial sum memoization for SimRank computation [LVGT10] can be applied to P-Rank in a similar way, but this still needs $O(Kn^3)$ time to compute P-Rank. For approximating P-Rank, Zhao *et al.* [ZHS09] have proposed the radius- or category-based pruning techniques to improve the estimation of P-Rank to $O(Kd^2n^2)$ worst-case time, with d being the average degree in a graph. However, this method is inherently heuristic, and even worse no theoretical guarantee is provided for the approximation error of the pruning results. Fortunately, we have an observation that a large body of vertices in a real network usually share some similar neighborhood structures (*e.g.,* similar user preference in a recommender system). Thus, we have an opportunity to "merge" these similar vertices, and devise fast algorithms to speed up P-Rank computation with accuracy guarantees.

4.1.2 Chapter Outlines

In this chapter, we consider the aforementioned P-Rank problems, encompassing the accuracy, stability and computational efficiency. The main results are the following.

- We provide an accuracy estimation for P-Rank iteration (Section 4.3). We show that $K = \lceil \ln \epsilon / \ln (\lambda \cdot C_{\text{in}} + (1-\lambda) \cdot C_{\text{out}}) \rceil$ iterations suffices to guarantee a desired accuracy ϵ, where λ is the weight factor and C_{in} and C_{out} are in- and out-link damping factors, respectively.

- We introduce the notion of *P-Rank condition number* κ_∞ to analyze the stability of P-Rank (Section 4.4). We deploy a new eigenvector-based approach to obtain a tight bound for κ_∞, and provide the conditions under which P-Rank is *stable*, that is, slight perturbations in the link structure will not cause large changes in the P-Rank similarity.

- We propose two novel matrix-based algorithms (DE P-Rank and UN P-Rank) for efficiently computing P-Rank in $O(r^4n^2 + r^2n)$ and $O(rn^2)$ time ($r \ll n$), respectively, over digraphs and undirected graphs (Section 4.5), as opposed to the conventional counterpart of $O(Kn^3)$ time. An error estimation is also provided as a by-product when a target low-rank υ ($\upsilon \ll r$) approximation is used for P-Rank.

- We empirically verify the efficiency of our methods on real and synthetic data (Section 4.6). The experimental results show that (1) P-Rank converges exponentially *w.r.t.* the iteration number; (2) the stability of P-Rank is sensitive to different choices of the damping factors and the weighted factor; (3) the proposed DE P-Rank and UN P-Rank outperform its competitors by almost one order of magnitude.

Symbol	Definition	Symbol	Definition
$\mathcal{G}$	network	r	rank of graph adjacency matrix ($r \ll n$)
$\mathcal{I}(a)$	in-neighbors of vertex a	υ	low rank of P-Rank approximation ($\upsilon \leq r$)
$\mathcal{O}(a)$	out-neighbors of vertex a	$s(a,b)$	P-Rank score between vertices a and b
n	number of vertices in $\mathcal{G}$	C_{in} / C_{out}	in-link / out-link damping factor
m	number of edges in $\mathcal{G}$	$\mathbf{A}$	adjacency matrix of $\mathcal{G}$
K	number of iterations	$\mathbf{S}$	P-Rank similarity matrix of $\mathcal{G}$
λ	weighting factor	$\mathbf{I}$	identity matrix

Table 4.1: Glossary of Symbols

4.2 Preliminaries

In accordance with [ZHS09], we assume that graphs studied in this paper have no multiple edges (corresponding to a 0-1 adjacency matrix). Table 4.1 lists the notations used throughout this chapter.

The basic essence underlying the P-Rank model [ZHS09] involves the following three facets:

1) Two distinct entities are similar if they are referenced by similar entities. (*In-link Recursion*)
2) Two distinct entities are similar if they reference similar entities. (*Out-link Recursion*)
3) Every entity is maximally similar to itself. (*Base Case*)

P-Rank Similarity. More formally, we revisit the formulation of P-Rank [ZHS09]. Given a network $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$. the P-Rank model can be formulated as follows:

For every two distinct vertices $u, v \in \mathcal{V}$, the similarity $s(u,v) \in [0,1]$ defined as

$$s(u,u) = 1; \tag{4.1}$$

$$s(u,v) = \overbrace{\frac{\lambda \cdot C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \sum_{i=1}^{|\mathcal{I}(u)|} \sum_{j=1}^{|\mathcal{I}(v)|} s(\mathcal{I}_i(u), \mathcal{I}_j(v))}^{\text{in-link part}} + \underbrace{\frac{(1-\lambda) \cdot C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \sum_{i=1}^{|\mathcal{O}(u)|} \sum_{j=1}^{|\mathcal{O}(v)|} s(\mathcal{O}_i(u), \mathcal{O}_j(v))}_{\text{out-link part}}, \tag{4.2}$$

is called *the P-Rank similarity* between u and v, where (i) $\lambda \in [0,1]$ is a weight factor, balancing the importance between in-links and out-links; (ii) C_{in} and $C_{\text{out}} \in (0,1)$ are the damping factors for in- and out-link directions, respectively; (iii) $\mathcal{I}(u)$ and $\mathcal{O}(u)$ are the in- and out-neighbor set of vertex u, respectively, with $\mathcal{I}_i(u)$ and $\mathcal{O}_i(u)$ being the i-th elements of $\mathcal{I}(u)$ and $\mathcal{O}(u)$, respectively; and (iv) $|\mathcal{I}(u)|$ and $|\mathcal{O}(u)|$ are the cardinalities of $\mathcal{I}(u)$ and $\mathcal{O}(u)$, respectively.

To avoid $s(u,v) = \infty$ in Eq.(4.2), we assume that

1) in-link part of Eq.(4.2)$= 0$ if $\mathcal{I}(u) = \varnothing$ or $\mathcal{I}(v) = \varnothing$;

2) out-link part of Eq.(4.2)$= 0$ if $\mathcal{O}(u) = \varnothing$ or $\mathcal{O}(v) = \varnothing$.

P-Rank Iterative Paradigm. The conventional method iteratively computes $s(u,v)$ as follows:

$$s^{(0)}(u,v) = \begin{cases} 0, \text{if } u \neq v; \\ 1, \text{if } u = v. \end{cases} \tag{4.3}$$

For each iteration $k = 1, 2, \cdots$, the k-th iterative P-Rank similarity $s^{(k)}(u,v)$ is iteratively computed as

$$\begin{aligned} s^{(k)}(u,u) &= 1; \\ s^{(k)}(u,v) &= \frac{\lambda \cdot C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \sum_{i=1}^{|\mathcal{I}(u)|} \sum_{j=1}^{|\mathcal{I}(v)|} s^{(k-1)}\left(\mathcal{I}_i(u), \mathcal{I}_j(v)\right) \\ &\quad + \frac{(1-\lambda) \cdot C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \sum_{i=1}^{|\mathcal{O}(u)|} \sum_{j=1}^{|\mathcal{O}(v)|} s^{(k-1)}\left(\mathcal{O}_i(u), \mathcal{O}_j(v)\right); \\ s^{(k)}(u,v) &= \text{Eq.(4.4)'s in-link part, if } \mathcal{O}(u) = \varnothing \text{ or } \mathcal{O}(v) = \varnothing; \\ s^{(k)}(u,v) &= \text{Eq.(4.4)'s out-link part, if } \mathcal{I}(u) = \varnothing \text{ or } \mathcal{I}(v) = \varnothing. \end{aligned} \tag{4.4}$$

It was proved in [ZHS09] that the sequence $\{s^{(k)}(u,v)\}$ non-decreasingly converges to the exact similarity $s(u,v)$, *i.e.*,

$$\lim_{k \to \infty} s^{(k)}(u,v) = s(u,v) \quad (\forall u, v \in \mathcal{V}). \tag{4.5}$$

4.3 P-Rank Accuracy Estimate

Despite the convergence of the sequence $\{s^{(k)}(u,v)\}_{k=0}^{\infty}$, the gap between the k-th iterative similarity $s^{(k)}(u,v)$ and the exact one $s(u,v)$ still remains unknown. This motivates us to study *the P-Rank accuracy estimate problem*:

Given a network $\mathcal{G}$, for each iteration number $k = 1, 2, \cdots$, it is to find a tight bound ϵ_k for the difference between the k-th iterative similarity $s^{(k)}(u,v)$ and the exact one $s(u,v)$ for $\forall u, v \in \mathcal{G}$.

The main result of this section is the following.

Theorem 4.1. *The P-Rank accuracy estimate problem has a tight upper bound*

$$\epsilon_k = (\lambda C_{in} + (1-\lambda) C_{out})^{k+1}$$

such that $\forall k = 0, 1, \cdots, \quad \forall u, v \in \mathcal{V}$,

$$|s(u,v) - s^{(k)}(u,v)| \leq \epsilon_k. \quad \square \tag{4.6}$$

Proof. (i) For $u = v$, Eq.(4.6) obviously holds since

$$s(u,u) - s^{(k)}(u,u) = 1 - 1 = 0. \quad (\forall k \geq 0, \ \forall u \in \mathcal{V})$$

(ii) For $u \neq v$, we use induction on k to prove Eq.(4.6) as follows:

Inductive Basis. We show that Eq.(4.6) holds for $k = 0$. Using Eq.(4.2) and $s^{(0)}(u,v) = 0$, we have

$$\begin{aligned}
& s(u,v) - s^{(0)}(u,v) = s(u,v) \\
&= \frac{\lambda \cdot C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \sum_{i=1}^{|\mathcal{I}(u)|} \sum_{j=1}^{|\mathcal{I}(v)|} \underbrace{s(\mathcal{I}_i(u), \mathcal{I}_j(v))}_{\leq 1} + \frac{(1-\lambda) \cdot C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \sum_{i=1}^{|\mathcal{O}(u)|} \sum_{j=1}^{|\mathcal{O}(v)|} \underbrace{s(\mathcal{O}_i(u), \mathcal{O}_j(v))}_{\leq 1} \\
&\leq \frac{\lambda \cdot C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \sum_{i=1}^{|\mathcal{I}(u)|} \sum_{j=1}^{|\mathcal{I}(v)|} 1 + \frac{(1-\lambda) \cdot C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \sum_{i=1}^{|\mathcal{O}(u)|} \sum_{j=1}^{|\mathcal{O}(v)|} 1 \\
&= \lambda \cdot C_{\text{in}} + (1-\lambda) \cdot C_{\text{out}}.
\end{aligned}$$

Inductive Step. Assume that Eq.(4.6) holds for a fixed k as the inductive hypothesis. We need to prove that Eq.(4.6) holds for $k+1$ as well. Combing Eqs.(4.2) and (4.4)

yields

$$
\begin{aligned}
& s(u,v) - s^{(k+1)}(u,v) \\
= & \frac{\lambda C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \sum_{i=1}^{|\mathcal{I}(u)|}\sum_{j=1}^{|\mathcal{I}(v)|} \overbrace{\left(s\left(\mathcal{I}_i(u),\mathcal{I}_j(v)\right) - s^{(k)}\left(\mathcal{I}_i(u),\mathcal{I}_j(v)\right)\right)}^{\text{using inductive hypothesis } \leq (\lambda C_{\text{in}}+(1-\lambda)C_{\text{out}})^{k+1}} \\
+ & \frac{(1-\lambda)\,C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \sum_{i=1}^{|\mathcal{O}(u)|}\sum_{j=1}^{|\mathcal{O}(v)|} \underbrace{\left(s\left(\mathcal{O}_i(u),\mathcal{O}_j(v)\right) - s^{(k)}\left(\mathcal{O}_i(u),\mathcal{O}_j(v)\right)\right)}_{\text{using inductive hypothesis } \leq (\lambda C_{\text{in}}+(1-\lambda)C_{\text{out}})^{k+1}} \\
\leq & \frac{\lambda C_{\text{in}}}{|\mathcal{I}(u)|\,|\mathcal{I}(v)|} \sum_{i=1}^{|\mathcal{I}(u)|}\sum_{j=1}^{|\mathcal{I}(v)|} \left(\lambda C_{\text{in}} + (1-\lambda)\,C_{\text{out}}\right)^{k+1} \\
& + \frac{(1-\lambda)\,C_{\text{out}}}{|\mathcal{O}(u)|\,|\mathcal{O}(v)|} \sum_{i=1}^{|\mathcal{O}(u)|}\sum_{j=1}^{|\mathcal{O}(v)|} \left(\lambda C_{\text{in}} + (1-\lambda)\,C_{\text{out}}\right)^{k+1} \\
= & \lambda C_{\text{in}}(\lambda C_{\text{in}} + (1-\lambda)\,C_{\text{out}})^{k+1} \\
& + (1-\lambda)\,C_{\text{out}}(\lambda C_{\text{in}} + (1-\lambda)\,C_{\text{out}})^{k+1} \\
= & \left(\lambda C_{\text{in}} + (1-\lambda)\,C_{\text{out}}\right)^{k+2}.
\end{aligned}
$$

This completes the induction. □

Theorem 4.1 provides an a-priori estimate for the gap between the iterative and exact P-Rank. For each iteration, this gap merely hinges on the λ, C_{in} and C_{out}. To be precise, for guaranteeing high accuracy at each iteration, it follows from

$$\epsilon_k = \left(\lambda(C_{\text{in}} - C_{\text{out}}) + C_{\text{out}}\right)^{k+1}$$

that smaller choices of C_{in} and C_{out} (i) with a smaller λ if $C_{\text{in}} > C_{\text{out}}$, or (ii) with a larger λ if $C_{\text{in}} < C_{\text{out}}$, will result in a smaller ϵ_k, and are thus more preferable.

Example 4.2. Setting $C_{\text{in}} = 0.6, C_{\text{out}} = 0.4, \lambda = 0.3, k = 5$ produces the following high accuracy for P-Rank:

$$\epsilon_k = \left(0.3 \times 0.6 + (1-0.3) \times 0.4\right)^{5+1} = 0.0095. \qquad \square$$

Note that the upper bound in Eq.(4.6) can be attainable. Consider the network $\mathcal{G}_0$ in Figure 4.1. It is apparent that $s^{(0)}(u,v) = 0$. For $k = 1, 2, \cdots$, it can be easily obtained

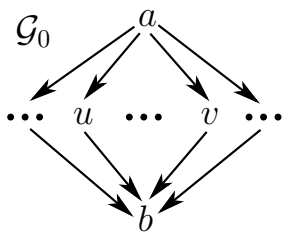


Figure 4.1: The equality of Eq.(4.6) is attainable for $\mathcal{G}_0$

that $s^{(k)}(u,v) = \lambda C_{\text{in}} + (1-\lambda)C_{\text{out}}$, which implies that $s(u,v) = \lambda C_{\text{in}} + (1-\lambda)C_{\text{out}}$. Hence, in the case of $k = 0$,

$$|s(u,v) - s^{(k)}(u,v)| = (\lambda C_{\text{in}} + (1-\lambda)C_{\text{out}})^{0+1},$$

which gives the precise upper bound in Eq.(4.6).

As a special case when $\lambda = 1$, Eq.(4.6) reduces to the SimRank accuracy estimate problem [LVGT10]. From this perspective, P-Rank accuracy estimate problem is an extension of Proposition 1 in [LVGT10] by jointly considering both in- and out-links for similarity assessment.

Conversely, the exponential P-Rank convergence rate in Theorem 4.1 implies that the total iteration number K of iterations needed for attaining a desired accuracy ϵ is

$$K = \lceil \ln \epsilon / \ln (\lambda \cdot C_{\text{in}} + (1-\lambda) \cdot C_{\text{out}}) \rceil .$$

4.4 Stability Analysis

In this section, the stability issue of P-Rank is investigated for analyzing the sensitivity of P-Rank similarity to the perturbations on a graph. First, a closed-form of P-Rank solution is represented as a matrix formula (Subsect. 4.4.1). Then, a rigorous bound of the P-Rank condition number based on this matrix formula is provided (Subsect. 4.4.2).

4.4.1 Closed-form of P-Rank

Let us first introduce some notations. For a network $\mathcal{G}$ with $n = |\mathcal{V}|$ vertices, we denote by (i) $\mathbf{A} = (a_{i,j}) \in \mathbb{R}^{n \times n}$ the *adjacency matrix* of $\mathcal{G}$ whose entry $a_{i,j}$ is 1 if there exists

an edge from vertex i to j, and 0 otherwise; (ii) $\mathbf{S} = (s_{i,j}) \in \mathbb{R}^{n\times n}$ the *P-Rank similarity matrix* whose entry $s_{i,j}$ equals the P-Rank score $s(i,j)$ between vertices i and j; and (iii) $\mathbf{Q} = (q_{i,j}) \in \mathbb{R}^{n\times n}$ and $\mathbf{P} = (p_{i,j}) \in \mathbb{R}^{n\times n}$ the *one-step backward and forward transition probability matrix* of $\mathcal{G}$, respectively, whose entries defined as follows:

$$q_{i,j} \triangleq \begin{cases} a_{j,i}/\sum_{j=1}^{n} a_{j,i}, & \text{if } \mathcal{I}(i) \neq \varnothing; \\ 0, & \text{if } \mathcal{I}(i) = \varnothing. \end{cases} \quad p_{i,j} \triangleq \begin{cases} a_{i,j}/\sum_{j=1}^{n} a_{i,j}, & \text{if } \mathcal{O}(i) \neq \varnothing; \\ 0, & \text{if } \mathcal{O}(i) = \varnothing. \end{cases} \tag{4.7}$$

With the above notations, the P-Rank formulae (4.1) and (4.2) can be rewritten as [1]

$$\mathbf{S} = \lambda C_{\text{in}} \cdot \mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T + (1-\lambda) C_{\text{out}} \cdot \mathbf{P} \cdot \mathbf{S} \cdot \mathbf{P}^T + (1 - \lambda C_{\text{in}} - (1-\lambda) C_{\text{out}}) \cdot \mathbf{I}_n. \tag{4.8}$$

Dividing both sides of Eq.(4.8) by $(1 - \lambda C_{\text{in}} - (1-\lambda)C_{\text{out}})$ results in

$$\mathbf{S} = (1 - \lambda C_{\text{in}} - (1-\lambda) C_{\text{out}}) \cdot \mathbf{S}', \text{ with}$$

$$\mathbf{S}' = \lambda C_{\text{in}} \cdot \mathbf{Q} \cdot \mathbf{S}' \cdot \mathbf{Q}^T + (1-\lambda) C_{\text{out}} \cdot \mathbf{P} \cdot \mathbf{S}' \cdot \mathbf{P}^T + \mathbf{I}_n. \tag{4.9}$$

Comparing Eq.(4.8) with Eq.(4.9), we see that the coefficient $(1 - \lambda C_{\text{in}} - (1-\lambda)C_{\text{out}})$ of $\mathbf{I}_n$ in Eq.(4.8) merely contributes an overall multiplicative factor to P-Rank similarity. Hence, setting this coefficient to 1 in Eq.(4.8) still preserves the *relative* magnitude of the P-Rank score even though the diagonal entries of $\mathbf{S}$ in this scenario may not be equal to 1. [2]

We also introduce two useful operators [GL96, p.180]: (i) $\text{vec}(\mathbf{X}) \in \mathbb{R}^{n^2}$ is defined to be the *vectorization* of the matrix $\mathbf{X} \in \mathbb{R}^{n\times n}$ formed by stacking the columns of $\mathbf{X}$ into a single column vector. (ii) $\mathbf{A} \otimes \mathbf{B}$ is the *Kronecker product* of the matrices $\mathbf{A}$ and $\mathbf{B}$.

To represent the closed-form of $\mathbf{S}$, we now take vec on both sides of Eq.(4.9), and then apply the Kronecker property $\text{vec}(\mathbf{AXB}) = (\mathbf{B}^T \otimes \mathbf{A}) \cdot \text{vec}(\mathbf{X})$ [GL96, p.180] to obtain

$$\text{vec}(\mathbf{S}') = (1-\lambda) C_{\text{out}} (\mathbf{P} \otimes \mathbf{P}) \cdot \text{vec}(\mathbf{S}') + \lambda C_{\text{in}} (\mathbf{Q} \otimes \mathbf{Q}) \cdot \text{vec}(\mathbf{S}') + \text{vec}(\mathbf{I}_n).$$

[1] Although in this case the diagonal entries of $\mathbf{S}$ may not equal 1, $\mathbf{S}$ still remains diagonally dominant, which ensures that "every vertex is maximally similar to itself".

[2] In what follows, we shall base our techniques on the P-Rank matrix form of Eq.(4.9).

Rearranging the terms in the above equation produces

$$\mathbf{M} \cdot \mathrm{vec}(\mathbf{S}') = \mathrm{vec}(\mathbf{I}_n) \text{ with}$$

$$\mathbf{M} = \mathbf{I}_{n^2} - \lambda C_{\text{in}}(\mathbf{Q} \otimes \mathbf{Q}) - (1-\lambda) C_{\text{out}}(\mathbf{P} \otimes \mathbf{P}), \tag{4.10}$$

which is a *linear* matrix equation in nature.

To show the existence of $\mathbf{M}^{-1}$, the following lemma is introduced.

Lemma 4.3. The matrices $\mathbf{Q} \otimes \mathbf{Q}$ and $\mathbf{P} \otimes \mathbf{P}$ are both row sub-stochastic matrices.[3] □

Proof. It follows from Eq.(4.7) that for each row $i = 1, \cdots, n$, the sum of each row in $\mathbf{Q}$ and $\mathbf{P}$ is no greater than 1. Then, for each row i' of $\mathbf{Q} \otimes \mathbf{Q}$, we have

$$\sum_{k=1}^{n} (q_{i',k} \sum_{j=1}^{n} q_{i,j}) \leq \sum_{k=1}^{n} (q_{i',k} \times 1) \leq 1,$$

which indicates that $\mathbf{Q} \otimes \mathbf{Q}$ is a row sub-stochastic matrix. A similar proof holds for $\mathbf{P} \otimes \mathbf{P}$. □

Based on Lemma 4.3, it can be easily proved that $\mathbf{M}$ in Eq.(4.10) is a diagonally dominant matrix, implying that $\mathbf{M}$ is invertible. Hence, by pre-multiplying $\mathbf{M}^{-1}$ on both sides of Eq.(4.10), the closed-form solution of P-Rank can be represented as

$$\mathrm{vec}(\mathbf{S}) = (1 - \lambda C_{\text{in}} - (1-\lambda) C_{\text{out}}) \cdot \mathbf{M}^{-1} \cdot \mathrm{vec}(\mathbf{I}_n). \tag{4.11}$$

4.4.2 Condition Number of P-Rank

Based on the closed-form of P-Rank solution in Eq.(4.11), we next analyze the P-Rank stability. One complicated factor in P-Rank stability issue is to precisely bound its condition number κ_∞, which has important implications for the sensitivity of P-Rank computation.

Let us first formally define the *P-Rank condition number*.

[3] *A row sub-stochastic matrix* is a non-negative matrix with each row sum being no greater than 1.

Definition 4.4 (P-Rank condition number)**.** For a network $\mathcal{G}$, let $\mathbf{Q}$ and $\mathbf{P}$ be the one-step backward and forward transition matrix defined by Eq.(4.7), respectively, and let $\mathbf{M}$ be defined by Eq.(4.10). Then, the quantity

$$\kappa_\infty(\mathcal{G}) \triangleq \|\mathbf{M}\|_\infty \cdot \|\mathbf{M}^{-1}\|_\infty \tag{4.12}$$

is called the P-Rank condition number of $\mathcal{G}$.

Here, $\|\star\|_\infty$ denotes the ∞-norm that returns the maximum absolute row sum of the matrix. □

The condition number is introduced for analyzing P-Rank stability. More specifically, we show the following result.

Theorem 4.5. *For any graph $\mathcal{G}$, the P-Rank condition number has the following tight bound*

$$\kappa_\infty(\mathcal{G}) \leq \frac{1+\lambda\cdot C_{in}+(1-\lambda)\cdot C_{out}}{1-\lambda\cdot C_{in}-(1-\lambda)\cdot C_{out}}. \quad \square \tag{4.13}$$

To prove Theorem 4.5, the following lemmas are necessary.

Lemma 4.6. $\|\mathbf{M}\|_\infty$ has the following upper bound:

$$\|\mathbf{M}\|_\infty \leq 1+\lambda\cdot C_{\text{in}}+(1-\lambda)\cdot C_{\text{out}}. \quad \square \tag{4.14}$$

Proof. By definition, the diagonal (i,i)-entry of $\mathbf{Q}\otimes\mathbf{Q}$ equals $q_{i',i'}\times q_{i'',i''}$, where

$$i' = \lceil i/n \rceil \text{ and } i'' = [(i-1) \bmod n] + 1.$$

Then, taking ∞-norm on both sides of Eq.(4.10) yields

$$\begin{aligned}\|\mathbf{M}\|_\infty &= \overbrace{\|\mathbf{I}_{n^2}\|_\infty}^{=1} + \lambda\cdot C_{\text{in}}\cdot \overbrace{\|\mathbf{Q}\otimes\mathbf{Q}\|_\infty}^{\leq 1} + (1-\lambda)\cdot C_{\text{out}}\cdot \overbrace{\|\mathbf{P}\otimes\mathbf{P}\|_\infty}^{\leq 1} \\ &\leq 1+\lambda\cdot C_{\text{in}}+(1-\lambda)\cdot C_{\text{out}}.\end{aligned} \tag{4.15}$$

□

Lemma 4.7. $\left\|\mathbf{M}^{-1}\right\|_\infty$ has the following upper bound:

$$\left\|\mathbf{M}^{-1}\right\|_\infty \leq \frac{1}{1+\lambda\cdot C_{\text{in}}+(1-\lambda)\cdot C_{\text{out}}}. \quad \square \tag{4.16}$$

Proof. Let $\mathbb{1}_{n^2}$ be a vector of length n^2 with entries of all 1s, and $\mathbf{e}_i$ a unit vector of length n^2 with a 1 in the i-th entry and 0s in all others.

Since $\mathbf{Q} \otimes \mathbf{Q}$ and $\mathbf{P} \otimes \mathbf{P}$ are row sub-stochastic matrices, it follows from Eq.(4.10) that $\forall i = 1, 2, \cdots, n^2$,

$$\begin{aligned}&\|\mathbf{I}_{n^2} - \mathbf{M} + [1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}] \cdot \mathbb{1}_{n^2} \cdot \mathbf{e}_i^T\|_\infty \\ =& \|\lambda C_{\text{in}}(\mathbf{Q} \otimes \mathbf{Q}) + (1-\lambda) C_{\text{out}}(\mathbf{P} \otimes \mathbf{P}) + [1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}] \cdot \mathbb{1}_{n^2} \cdot \mathbf{e}_i^T\|_\infty \leq 1.\end{aligned}$$

Hence, $\mathbf{I}_{n^2} - \mathbf{M} + [1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}] \cdot \mathbb{1}_{n^2} \cdot \mathbf{e}_i^T$ is a row sub-stochastic matrix. Due to the spectral radius property $\rho(\star) \leq \|\star\|_\infty$, it follows that

$$\rho(\mathbf{I}_{n^2} - \mathbf{M} + [1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}] \cdot \mathbb{1}_{n^2} \cdot \mathbf{e}_i^T) \leq 1.$$

Notice that $\mathbf{I}_{n^2} - \mathbf{M} + [1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}] \cdot \mathbb{1}_{n^2} \cdot \mathbf{e}_i^T$ is nonnegative. According to *the eigen-pair property for the nonnegative matrix* [4], there exists some row-vector $\mathbf{x}_i^T$ with $\|\mathbf{x}_i^T\|_\infty = 1$ such that $\forall i = 1, 2, \cdots, n^2$,

$$\mathbf{x}_i^T \cdot (\mathbf{I}_{n^2} - \mathbf{M} + [1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}] \cdot \mathbb{1}_{n^2} \cdot \mathbf{e}_i^T) \leq \mathbf{x}_i^T.$$

Rearranging the terms in the above inequality produces

$$\mathbf{x}_i^T \cdot \mathbf{M} \geq [1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}] \cdot \mathbf{x}_i^T \cdot \mathbb{1}_{n^2} \cdot \mathbf{e}_i^T. \tag{4.17}$$

Note that $\|\mathbf{x}_i^T\|_\infty = 1$, which implies that $\mathbf{x}_i^T \cdot \mathbb{1}_{n^2} = 1$. Post-multiplying by $\mathbf{M}^{-1}$ on both sides of Eq.(4.17) produces $\forall i = 1, 2 \cdots, n^2$,

$$\mathbf{e}_i^T \cdot \mathbf{M}^{-1} \leq 1/(1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}) \cdot \mathbf{x}_i^T.$$

Also, notice that $\|\mathbf{M}^{-1}\|_\infty = \max_{1 \leq i \leq n^2} \|\mathbf{e}_i^T \cdot \mathbf{M}^{-1}\|_\infty$ and $\|\mathbf{x}_i^T\|_\infty = 1$. Taking ∞-norm on both sides of the above inequality yields Eq.(4.16). □

From Lemmas 4.6 and 4.7, Theorem 4.5 follows directly, which provides a tight bound for $\kappa_\infty(\mathcal{G})$. Intuitively, $\kappa_\infty(\mathcal{G})$ has two important implications. Foremost, it can evaluate

[4] According to [Mey01, p.670], for a nonnegative matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$, there exists a vector $\mathbf{x} \in \{\mathbf{z} | \mathbf{z} \geq \mathbf{0}$ with $\mathbf{z} \neq \mathbf{0}\}$ such that $\mathbf{A}\mathbf{x} = \rho(\mathbf{A})\mathbf{x}$.

how stable the P-Rank similarity is to the perturbations in the link structure of graph $\mathcal{G}$ (by inserting or deleting vertices or edges, or by changing the value of weighted factor λ and damping factors C_{in} and C_{out}). Moreover, it can estimate the error of P-Rank ranking results invoked by the roundoff error in P-Rank iterations.

To get a feel for how $\kappa_\infty(\mathcal{G})$ affects P-Rank stability, we denote by $\Delta\mathbf{M}$ the updates to the original matrix $\mathbf{M}$ defined in Eq.(4.10), and $\Delta\mathbf{S}$ the updates to the original similarity matrix $\mathbf{S}$. From the closed-form solution of P-Rank, it is known that

$$\begin{aligned}\frac{\|\Delta\mathbf{S}\|_{\max}}{\|\mathbf{S}\|_{\max}} = \frac{\|\text{vec}(\Delta\mathbf{S})\|_\infty}{\|\text{vec}(\mathbf{S})\|_\infty} &\leq \kappa_\infty(\mathcal{G}) \cdot \frac{\|\Delta\mathbf{M}\|_\infty}{\|\mathbf{M}\|_\infty} \\ &\leq \frac{1+\lambda\cdot C_{\text{in}}+(1-\lambda)\cdot C_{\text{out}}}{1-\lambda\cdot C_{\text{in}}-(1-\lambda)\cdot C_{\text{out}}} \cdot \frac{\|\Delta\mathbf{M}\|_\infty}{\|\mathbf{M}\|_\infty},\end{aligned}$$

where $\|\mathbf{X}\|_{\max} = \max_{1\leq i,j\leq n}\{x_{i,j}\}$ is a maximum elementwise matrix norm. This tells that smaller $\kappa_\infty(\mathcal{G})$ (*i.e.,* smaller choices of C_{in} and C_{out}) makes P-Rank more stable, implying that a small change $\Delta\mathbf{M}$ in the link structure to $\mathbf{M}$ may not cause a large change $\Delta\mathbf{S}$ in P-Rank similarity scores. Conversely, the larger value of $\kappa_\infty(\mathcal{G})$ makes P-Rank *ill-conditioned*. [5]

The P-Rank condition number $\kappa_\infty(\mathcal{G})$ can vary with the choice of weighting factor λ. To see this, let us compute the partial derivatives *w.r.t.* λ in Eq.(4.13) :

$$\frac{\partial}{\partial\lambda}\left(\frac{1+\lambda\cdot C_{\text{in}}+(1-\lambda)\cdot C_{\text{out}}}{1-\lambda\cdot C_{\text{in}}-(1-\lambda)\cdot C_{\text{out}}}\right) = \frac{2\,(C_{\text{in}}-C_{\text{out}})}{(1-\lambda\cdot C_{\text{in}}-(1-\lambda)\cdot C_{\text{out}})^2}. \tag{4.18}$$

This implies that when $C_{\text{in}} > C_{\text{out}}$ (*resp.*$C_{\text{in}} < C_{\text{out}}$), for the increased λ, a small change in $\mathcal{G}$ may result in a large (*resp.*small) change in P-Rank, which makes P-Rank an *ill-conditioned* (*resp.a well-conditioned*) problem; when $C_{\text{in}} = C_{\text{out}}$, the value of $\kappa_\infty(\mathcal{G})$ is independent of λ.

It is worth noting that the upper bound of $\kappa_\infty(\mathcal{G})$ in Eq.(4.13) is attainable if and only if each vertex in network $\mathcal{G}$ has at least one in-degree and one out-degree because in this case each row sum and each column sum of $\mathbf{A}$ are *strictly* greater than 0, which

[5] A detailed discussion will be given in Subsection 4.5.1 for measuring the effectiveness of P-Rank approximate algorithm to further appreciate the utility of P-Rank condition number $\kappa_\infty(\mathcal{G})$.

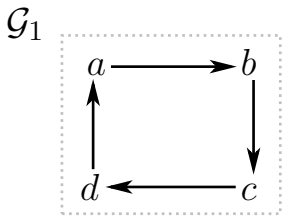


Figure 4.2: The equality of Eq.(4.13) is attainable for $\mathcal{G}_1$

ensures that $\mathbf{Q}$ and $\mathbf{P}$ are exactly row stochastic matrices [6] rather than sub-stochastic ones, and hence $\|\mathbf{Q} \otimes \mathbf{Q}\|_\infty = \|\mathbf{P} \otimes \mathbf{P}\|_\infty = 1$.

Example 4.8. Consider a directed cycle network of length 4, depicted as $\mathcal{G}_1$ in Figure 4.2, in which each vertex has one in-link and one out-link. Setting $\lambda = 0.5$, $C_{\text{in}} = 0.8$, and $C_{\text{out}} = 0.6$, one can verify that the equality of Eq.(4.13) is attained for $\mathcal{G}_1$ as follows.

On one hand, since $\mathbf{A} = \mathbf{Q} = \mathbf{P}$ for $\mathcal{G}_1, \mathbf{M}$ and $\mathbf{M}^{-1}$ can be solved naively from Eq.(4.10), which follows that

$$\kappa_\infty(\mathcal{G}) = \|\mathbf{M}\|_\infty \cdot \|\mathbf{M}^{-1}\|_\infty \doteq 1.7 \times 3.333 = 5.667;$$

on the other, computing the right-hand side of (4.13) produces

$$\frac{1 + \lambda \cdot C_{\text{in}} + (1-\lambda) \cdot C_{\text{out}}}{1 - \lambda \cdot C_{\text{in}} - (1-\lambda) \cdot C_{\text{out}}} = \frac{1 + 0.5 \times 0.8 + (1 - 0.5) \times 0.6}{1 - 0.5 \times 0.8 - (1 - 0.5) \times 0.6} \doteq 5.667.$$

Both results are exactly the same, and hence the equality of Eq.(4.13) holds. □

4.5 Optimization Techniques

In this section, optimization techniques for P-Rank computation are suggested. (i) For directed networks, an efficient algorithm based on low rank approximation of the transition matrices is proposed for reducing the calculations of trivial similarity values associated with the small singular values (Subsection 4.5.1). (ii) For undirected networks, a P-Rank solution in terms of eigenfunctions is introduced for further optimizing similarity computations (Subsection 4.5.2).

[6] A matrix having row sums equal to 1 is called *a row stochastic matrix*.

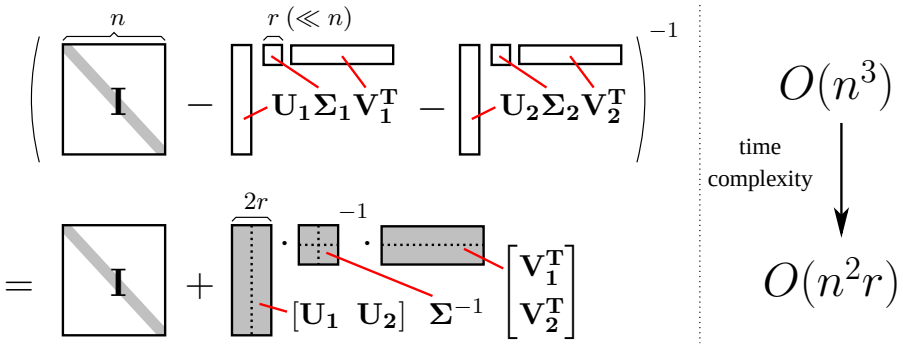


Figure 4.3: Low-rank update of matrix inversion

4.5.1 P-Rank on Digraphs

We next devise an algorithm for speeding up the computation of P-Rank to $O\left(r^4n^2 + r^2n\right)$ from $O(Kn^4)$ [ZHS09] time in the worst case, where r ($\ll n$) is the rank of graph adjacency matrix, and K is the total iteration number.

The main idea in optimizing P-Rank computation is to "merge" the vertices having similar neighbor structures by utilizing a singular value decomposition of a graph that is represented as a matrix inversion. To effectively decompose the graph, a low-rank update formula is also proposed to compute this matrix inversion in the $r \times r$ dimension rather than its conventional counterpart in $n \times n$ dimension. We observe that vertices in a real-world graph (*e.g.,* bibliographic networks [JW02], who-trusts-whom social networks [LHK10]) are often sparse and may share the similar structure of the neighborhood. Thus, the rank r of the adjacency matrix is typically much smaller than n, and the computational efficiency can be highly achieved.

For an elaborate discussion, we first establish the following low-rank update of the matrix inversion identity, which is useful to subsequent P-Rank optimization.

Lemma 4.9. Let $\mathbf{I}$ be an $n \times n$ identity matrix, $\mathbf{U}_i$ and $\mathbf{V}_i$ be $n \times r$ matrices , and $\mathbf{C}_i$ be $r \times r$ matrices $(i = 1, 2)$. Then the following *matrix inversion identity* holds.

$$(\mathbf{I} - \mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T - \mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T)^{-1} = \mathbf{I} + \begin{pmatrix}\mathbf{U}_1 \mathbf{U}_2\end{pmatrix} \begin{pmatrix} {\mathbf{\Sigma}_1}^{-1} - \mathbf{V}_1^T\mathbf{U}_1 & -\mathbf{V}_1^T\mathbf{U}_2 \\ -\mathbf{V}_2^T\mathbf{U}_1 & {\mathbf{\Sigma}_2}^{-1} - \mathbf{V}_2^T\mathbf{U}_2 \end{pmatrix}^{-1} \begin{pmatrix}\mathbf{V}_1^T \\ \mathbf{V}_2^T\end{pmatrix} \quad \square$$

Proof. Let $\mathbf{U} = (\mathbf{U}_1\, \mathbf{U}_2)$, $\mathbf{V} = (\mathbf{V}_1\, \mathbf{V}_2)$ and $\mathbf{\Sigma} = \begin{pmatrix}\mathbf{\Sigma}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{\Sigma}_2\end{pmatrix}$. Substituting in both sides of

the *Woodbury formula* [GL96] $\left(\mathbf{I}-\mathbf{U}\mathbf{C}\mathbf{V}^T\right)^{-1}=\mathbf{I}^{-1}+\mathbf{I}^{-1}\mathbf{U}\left(\mathbf{\Sigma}^{-1}-\mathbf{V}^T\mathbf{I}^{-1}\mathbf{U}\right)^{-1}\mathbf{V}^T\mathbf{I}^{-1}$ gives

$$\begin{aligned}
\text{LHS} &= \left(\mathbf{I}-\begin{pmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{pmatrix}\begin{pmatrix}\mathbf{\Sigma}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{\Sigma}_2\end{pmatrix}\begin{pmatrix}\mathbf{V}_1^T\\ \mathbf{V}_2^T\end{pmatrix}\right)^{-1}\\
&= \left(\mathbf{I}-\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T-\mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T\right)^{-1},\\
\text{RHS} &= \mathbf{I}+\begin{pmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{pmatrix}\left(\begin{pmatrix}\mathbf{\Sigma}_1 & \mathbf{0}\\ \mathbf{0} & \mathbf{\Sigma}_2\end{pmatrix}^{-1}-\begin{pmatrix}\mathbf{V}_1^T\\ \mathbf{V}_2^T\end{pmatrix}\begin{pmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{pmatrix}\right)^{-1}\begin{pmatrix}\mathbf{V}_1^T\\ \mathbf{V}_2^T\end{pmatrix}\\
&= \mathbf{I}+\begin{pmatrix}\mathbf{U}_1 & \mathbf{U}_2\end{pmatrix}\begin{pmatrix}{\mathbf{\Sigma}_1}^{-1}-\mathbf{V}_1^T\mathbf{U}_1 & -\mathbf{V}_1^T\mathbf{U}_2\\ -\mathbf{V}_2^T\mathbf{U}_1 & {\mathbf{\Sigma}_2}^{-1}-\mathbf{V}_2^T\mathbf{U}_2\end{pmatrix}^{-1}\begin{pmatrix}\mathbf{V}_1^T\\ \mathbf{V}_2^T\end{pmatrix}.
\end{aligned}$$

Since LHS=RHS, this yields the desired results. □

A consequence of Lemma 4.9 is to convert an $n\times n$ matrix inversion into an $r\times r$ matrix inversion with $r\ll n$, thus greatly improving the time complexity. More precisely, as depicted in Figure 4.3, when one wishes to compute the matrix inversion $(\mathbf{I}-\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T-\mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T)^{-1}$, it is only necessary to calculate the RHS of Eq.(4.19), which is in $O(n^2r+r^2n+r^3)$ $(r\ll n)$ due to the low-rank of $\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T$ and $\mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T$, as opposed to the naive $O\left(n^3\right)$ time of matrix inversion [HJ90].

Lemma 4.9 is established in conjunction with singular value decomposition for optimizing P-Rank computation. The key observation is that $\mathbf{Q}$ and $\mathbf{P}$ may have small matrix rank $r(\ll n)$ as most real graphs are often sparse. [7] Comparing the LHS of Eq.(4.19) with the closed-form of P-Rank solution (Eq.(4.11)), it can be noticed that once the matrix $\lambda C_{\text{in}}\left(\mathbf{Q}\otimes\mathbf{Q}\right)$ and $(1-\lambda)\,C_{\text{out}}\left(\mathbf{P}\otimes\mathbf{P}\right)$ are decomposed into the form of $\mathbf{U}_1\mathbf{\Sigma}_1\mathbf{V}_1^T$ and $\mathbf{U}_2\mathbf{\Sigma}_2\mathbf{V}_2^T$, respectively, Lemma 4.9 can be used for speeding up the computation of $\mathbf{M}^{-1}$ in Eq.(4.11) .

More specifically, we show the following main result in this subsection.

Theorem 4.10. *Let r be the rank of the graph adjacency matrix. Given a low rank*

[7] A sparse graph $\mathcal{G}=(\mathcal{V},\mathcal{E})$ is a graph with $|\mathcal{E}|=O(|\mathcal{V}|)$.

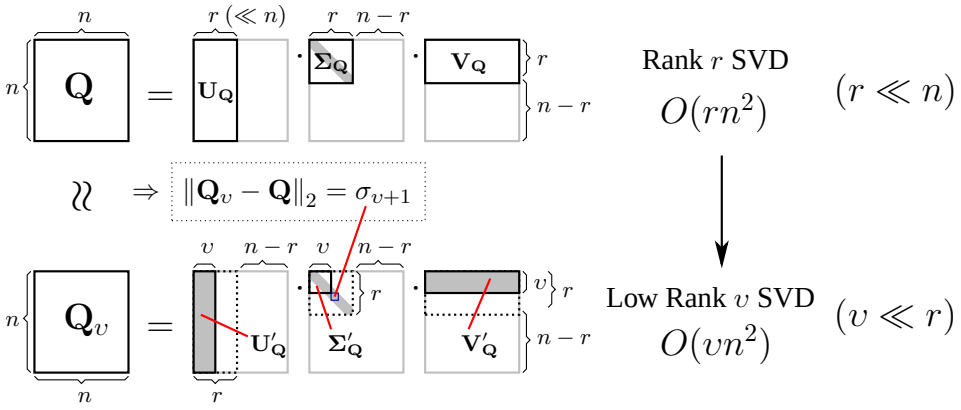


Figure 4.4: Low rank υ approximation truncating the smallest $r-\upsilon$ singular values of $\mathbf{Q}$

υ $(\leq r)$, *it is in* $O\left(\upsilon^4 n^2 + \upsilon^2 n\right)$ *time to estimate P-Rank up to an additive error*

$$\epsilon_\upsilon \leq \frac{\lambda C_{in}\sigma_1\sigma_{\upsilon+1}r + (1-\lambda)\, C_{out}\bar{\sigma}_1\bar{\sigma}_{\upsilon+1}r}{1-\lambda C_{in} - (1-\lambda)\, C_{out}},$$

where σ_i *and* $\bar{\sigma}_i$ $(i = 1, \upsilon+1)$ *are the* i*-th largest singular values of* $\mathbf{Q}$ *and* $\mathbf{P}$ *respectively.*

□

(A detailed proof will be provided after some discussions.)

In particular, setting the low-order υ to r (the rank of adjacency matrix) gives the following corollary.

Corollary 4.11. For digraphs, the exact P-Rank similarity $\mathbf{S}$ in Eq.(4.8) is solvable in $O(r^4n^2 + r^2n)$ time. □

As remarked earlier, it takes $O\left(n^2r + r^2n\right)$ time to perform the matrix inversion in Lemma 4.9. This tells us that P-Rank estimation does not make our lives much harder since $\upsilon(\leq r)$ is typically much smaller than n in practice. In contrast with the *quartic* time of its traditional counterpart via an iterative paradigm [ZHS09], the low-rank υ $(\ll r)$ approximation of P-Rank allows us to estimate similarities in *quadratic* time in the number of vertices.

The a-posteriori error ϵ_υ in Theorem 4.10 is acceptable in practice (*e.g.,* WIKIand DBLP) since the $(\upsilon+1)$-th largest singular values $\sigma_{\upsilon+1}$ and $\bar{\sigma}_{\upsilon+1}$ are reasonably small. As depicted in Figure 4.4, the low-rank υ $(\leq r)$ decomposition procedure truncates the

smallest $r-\upsilon$ almost zero singular values of the adjacency matrix which (1) contain little practical information for computing the resultant similarity, and (2) require considerable amounts of space for subsequent computations. For instance, setting $C_{\text{in}} = C_{\text{out}} = 0.8$ and $\lambda = 0.5$ (as suggested in [ZHS09]) implies a high accuracy

$$\epsilon_\upsilon \leq \frac{0.5 \times 0.8 \times 1.12 + 0.5 \times 0.8 \times 1.08}{1 - 0.5 \times 0.8 - 0.5 \times 0.8} \times 10^{-7} \times 15\text{K} \approx 0.006,$$

for an English WIKIgraph of 1.2M vertices ($\upsilon = r/2 \approx 15\text{K}$, $\sigma_1 = 1.12$, $\bar{\sigma}_1 = 1.08$) with $\max(\sigma_{\upsilon+1}, \bar{\sigma}_{\upsilon+1}) < 10^{-7}$ being truncated.

The choice of low rank υ ($\leq r$) has a user-controlled effect over the approximation error, which is a time-accuracy trade-off. As an extreme case of $\upsilon = r$ ($\ll n$), we notice that $\sigma_{r+1} = \bar{\sigma}_{r+1} = 0$ and the approximate P-Rank similarity becomes the exact P-Rank similarity. From this perspective, the approximate P-Rank is an extension of the conventional exact approach.

We next prove Theorem 4.10 by providing an algorithm for the low-rank υ P-Rank approximation on digraphs.

Algorithm. The algorithm, referred to as DE P-Rank, is shown in Algorithm 4.1. It accepts as input a web graph $\mathcal{G}$, a weighted factor λ, two damping factors C_{in} and C_{out}, and a low rank υ (an optional parameter). If υ is omitted, the default value is the rank r of adjacency matrix in $\mathcal{G}$. The algorithm returns the approximate P-Rank similarity matrix $\mathbf{S}$ for $\mathcal{G}$ and an accuracy ϵ_υ if $\upsilon \leq r$; otherwise it returns the exact similarity $\mathbf{S}$ for $\epsilon_\upsilon = 0$.

Before illustrating the algorithm, we first present the notations it uses. (1) `RowNorm` ($\mathbf{A}$) returns a row-stochastic matrix by normalizing each nonzero row of $\mathbf{A}$; `Rank` ($\mathbf{A}$) returns the rank of $\mathbf{A}$. (2) Given a matrix $\mathbf{Q}$ and a positive integer υ, `RSVD` ($\mathbf{Q}, \upsilon$) returns the low-rank υ singular matrix decomposition $\mathbf{Q}_\upsilon = \mathbf{U}_\mathbf{Q}\mathbf{\Sigma}_\mathbf{Q}\mathbf{V}_\mathbf{Q}^T$ that minimizes $\|\mathbf{Q}_\upsilon - \mathbf{Q}\|_2 = \sigma_{\upsilon+1}$, where $\mathbf{U}_\mathbf{Q}$ and $\mathbf{V}_\mathbf{Q}$ are $n \times \upsilon$ orthogonal matrices, and $\mathbf{\Sigma}_\mathbf{Q}$ is a $\upsilon \times \upsilon$ diagonal matrix. (3) We use $\tilde{\mathbf{U}}_\mathbf{Q}, \tilde{\mathbf{\Sigma}}_\mathbf{Q}, \tilde{\mathbf{V}}_\mathbf{Q}^T$ to denote the self-Kronecker products

Algorithm 4.1: DE P-Rank $(\mathcal{G}, \lambda, C_{\text{in}}, C_{\text{out}}, \upsilon)$

Input : web graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, weight factor λ,

damping factors C_{in} and C_{out}, low rank υ.

Output: similarity matrix $\mathbf{S}$, and accuracy ϵ_υ.

initialize the adjacency matrix $\mathbf{A}$ of $\mathcal{G}$.

compute the transition matrices $\mathbf{Q}$ and $\mathbf{P}$ in $\mathcal{G}$:

$\mathbf{Q} \leftarrow \texttt{RowNorm}\left(\mathbf{A}^T\right)$, $\mathbf{P} \leftarrow \texttt{RowNorm}\left(\mathbf{A}\right)$.

if υ *is empty* **then** $\upsilon \leftarrow \texttt{Rank}\left(\mathbf{A}\right)$

do low rank SVD approximation for $\mathbf{Q}$ and $\mathbf{P}$:

$[\mathbf{U_Q}, \mathbf{\Sigma_Q}, \mathbf{V_Q}; \sigma_1, \sigma_{\upsilon+1}] \leftarrow \texttt{RSVD}\ (\mathbf{Q}, \upsilon)$,

$[\mathbf{U_P}, \mathbf{\Sigma_P}, \mathbf{V_P}; \bar{\sigma}_1, \bar{\sigma}_{\upsilon+1}] \leftarrow \texttt{RSVD}\ (\mathbf{P}, \upsilon)$.

compute the self-Kronecker products :

$$\tilde{\mathbf{U}}_\mathbf{Q} \leftarrow \mathbf{U_Q} \otimes \mathbf{U_Q}, \quad \tilde{\mathbf{\Sigma}}_\mathbf{Q} \leftarrow \mathbf{\Sigma_Q} \otimes \mathbf{\Sigma_Q}, \quad \tilde{\mathbf{V}}_\mathbf{Q} \leftarrow \mathbf{V_Q} \otimes \mathbf{V_Q},$$

$$\tilde{\mathbf{U}}_\mathbf{P} \leftarrow \mathbf{U_P} \otimes \mathbf{U_P}, \quad \tilde{\mathbf{\Sigma}}_\mathbf{P} \leftarrow \mathbf{\Sigma_P} \otimes \mathbf{\Sigma_P}, \quad \tilde{\mathbf{V}}_\mathbf{P} \leftarrow \mathbf{V_P} \otimes \mathbf{V_P}.$$

compute the matrix $\mathbf{\Sigma}$:

$$\mathbf{\Sigma}_{11} \leftarrow \tfrac{1}{\lambda C_{\text{in}}} \tilde{\mathbf{\Sigma}}_\mathbf{Q}^{-1} - \tilde{\mathbf{V}}_\mathbf{Q}^T \tilde{\mathbf{U}}_\mathbf{Q}, \quad \mathbf{\Sigma}_{12} \leftarrow \tilde{\mathbf{V}}_\mathbf{Q}^T \tilde{\mathbf{U}}_\mathbf{P},$$

$$\mathbf{\Sigma}_{22} \leftarrow \tfrac{1}{(1-\lambda) C_{\text{out}}} \tilde{\mathbf{\Sigma}}_\mathbf{P}^{-1} - \tilde{\mathbf{V}}_\mathbf{P}^T \tilde{\mathbf{U}}_\mathbf{P}, \quad \mathbf{\Sigma}_{21} \leftarrow -\tilde{\mathbf{V}}_\mathbf{P}^T \tilde{\mathbf{U}}_\mathbf{Q}.$$

compute the P-Rank similarity vector $\mathbf{s}$:

$$\mathbf{v}_1 \leftarrow \left(\tilde{\mathbf{V}}_\mathbf{Q}\ \tilde{\mathbf{V}}_\mathbf{P} \right)^T \text{vec}\left(\mathbf{I}_n\right), \quad \mathbf{v}_2 \leftarrow \begin{pmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{pmatrix}^{-1} \mathbf{v}_1$$

$$\mathbf{s} \leftarrow \text{vec}\left(\mathbf{I}_n\right) + \left(\tilde{\mathbf{U}}_\mathbf{Q}\ \tilde{\mathbf{U}}_\mathbf{P} \right) \mathbf{v}_2.$$

if $\upsilon < \texttt{Rank}\left(\mathbf{A}\right)$ **then** estimate accuracy

$$\epsilon_\upsilon \leftarrow \frac{\lambda C_{\text{in}} \sigma_1 \sigma_{\upsilon+1} + (1-\lambda)\, C_{\text{out}} \bar{\sigma}_1 \bar{\sigma}_{\upsilon+1}}{1 - \lambda C_{\text{in}} - (1-\lambda)\, C_{\text{out}}} \texttt{Rank}\left(\mathbf{A}\right)$$

else $\epsilon_\upsilon \leftarrow 0$.

reshape the $n \times n$ similarity matrix $\mathbf{S}$ *s.t.* $\mathbf{s} = \text{vec}\left(\mathbf{S}\right)$.

return $\mathbf{S}$ and ϵ_υ.

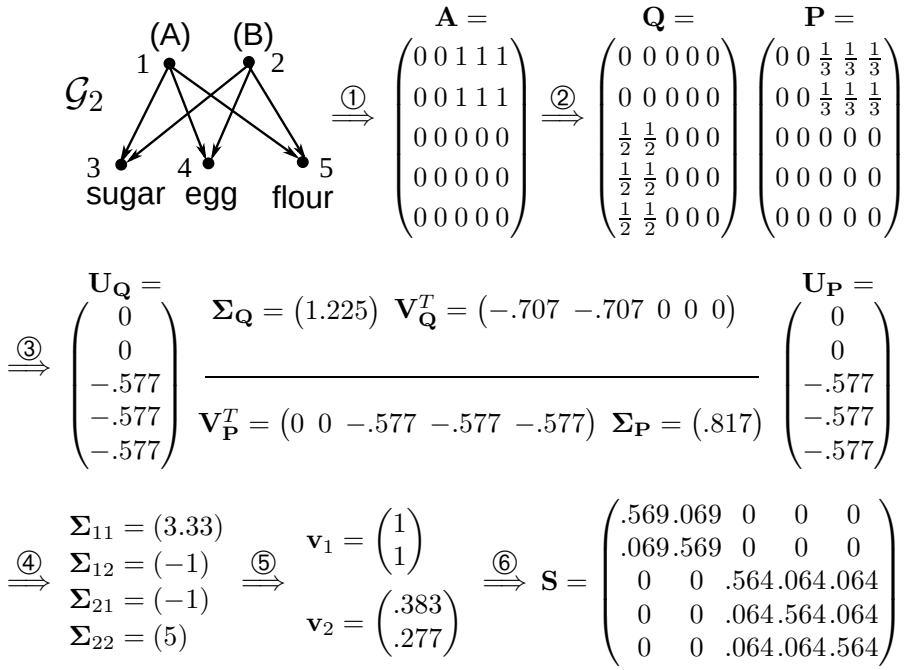


Figure 4.5: Heterogenous Shopping Graph $\mathcal{G}_2$

of $\mathbf{U_Q}, \mathbf{\Sigma_Q}, \mathbf{V_Q^T}$, respectively.

The algorithm DE P-Rank works as follows. It first initializes the adjacency matrix $\mathbf{A}$ of $\mathcal{G}$ (line 1). Using $\mathbf{A}$, it then computes $\mathbf{Q}$ and $\mathbf{P}$ by normalizing each nonzero row of $\mathbf{A}$ and $\mathbf{A}^T$, respectively (line 2). When the optional argument υ is not supplied, DE P-Rank also provides a default value $\texttt{Rank}(\mathbf{A})$ for υ (line 3).

For the matrix $\mathbf{Q}$ (*resp.*$\mathbf{P}$) and the rank υ, DE P-Rank then uses RSVD () to decompose the large $\mathbf{Q}$ (*resp.*$\mathbf{P}$) into the product of small matrices, *i.e.*, $\mathbf{U_Q}\mathbf{\Sigma_Q}\mathbf{V_Q^T}$ (*resp.*$\mathbf{U_P}\mathbf{\Sigma_P}\mathbf{V_P^T}$), which results in a more compact representation of $\mathcal{G}$ (line 5). In the case when $\upsilon <$ $\texttt{Rank}(\mathbf{A})$, RSVD () provides the best low-rank υ approximation of $\mathbf{Q}$ (*resp.*$\mathbf{P}$) in the least square error sense; otherwise, it yields exact factorizations for $\mathbf{Q}$ and $\mathbf{P}$. Moreover, DE P-Rank utilizes the matrices $\mathbf{U_Q}, \mathbf{\Sigma_Q}, \mathbf{V_Q^T}, \mathbf{U_P}, \mathbf{\Sigma_P}, \mathbf{V_P^T}$ to obtain the P-Rank similarity matrix $\mathbf{S}$ based on Lemma 4.9 (lines 5-7). More concretely, the block matrices $(\tilde{\mathbf{U}}_\mathbf{Q}\ \tilde{\mathbf{U}}_\mathbf{P})$, $\begin{pmatrix}\mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12}\\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22}\end{pmatrix}$ and $(\tilde{\mathbf{V}}_\mathbf{Q}\ \tilde{\mathbf{V}}_\mathbf{P})^T$ can be derived from the self-Kronecker products of $\mathbf{U_Q}, \mathbf{\Sigma_Q}, \mathbf{V_Q^T}, \mathbf{U_P}, \mathbf{\Sigma_P}, \mathbf{V_P^T}$ respectively when applied to compute vec($\mathbf{S}$) via matrix-vector multiplication.

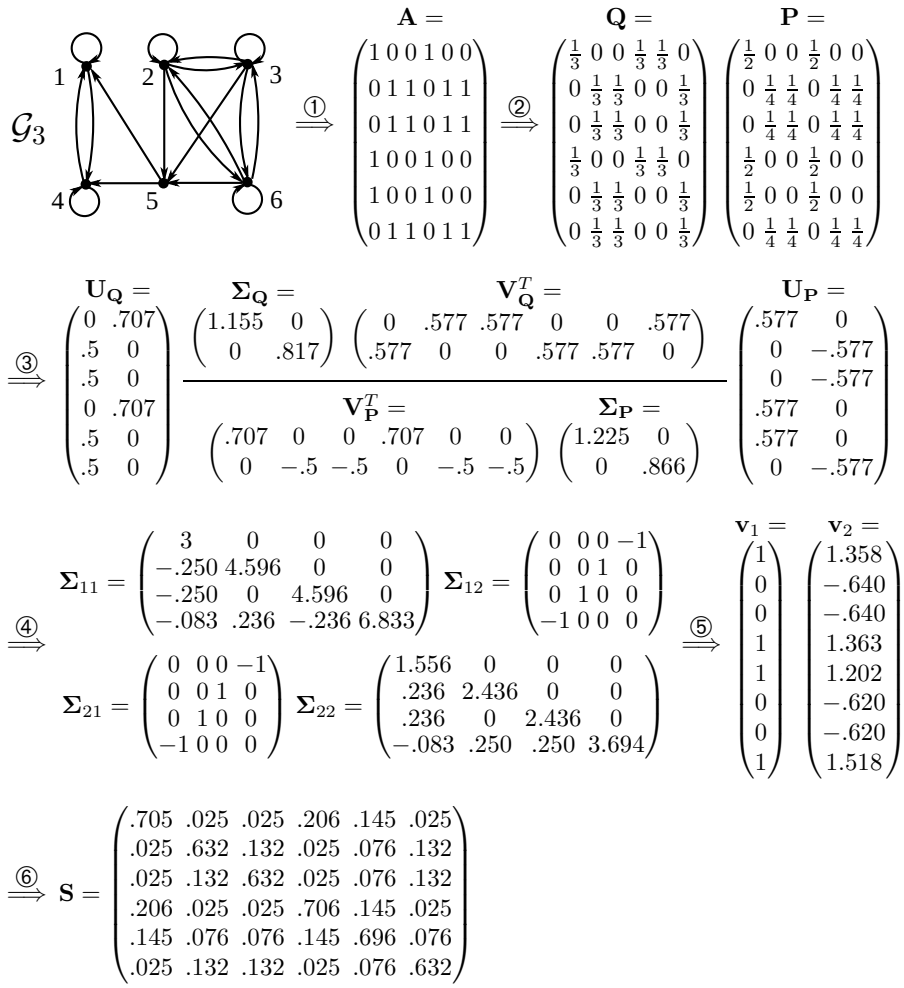


Figure 4.6: Homogeneous Scientific Paper Network $\mathcal{G}_3$

When $\mathbf{s}$ (*i.e.,* vec($\mathbf{S}$)) is derived, DE P-Rank compares the low rank υ with $\texttt{Rank}(\mathbf{A})$. If $\upsilon < \texttt{Rank}(\mathbf{A})$, the accuracy ϵ_υ also needs to be estimated for the best low-rank υ approximation (line 8). The entries of $\mathbf{s}$ are the desired P-Rank similarities reshaped in the matrix $\mathbf{S}$, which is returned as the final result (lines 9-10).

Example 4.12 (Heterogenous Graph)**.** Figure 4.5 depicts how DE P-Rank calculates similarity scores (using $C_{\text{in}} = 0.4$, $C_{\text{out}} = 0.6$, $\lambda = 0.5$) in a bipartite shopping graph $\mathcal{G}_2$ consisting of two types of entities: persons (A) and (B) purchased the same items sugar, egg, flour.

DE P-Rank first computes $\mathbf{Q}$ and $\mathbf{P}$ from the adjacency matrix $\mathbf{A}$ of $\mathcal{G}_2$. It then decomposes $\mathbf{Q}$ and $\mathbf{P}$ into $\mathbf{U_Q \Sigma_Q V_Q^T}$ and $\mathbf{U_P \Sigma_P V_P^T}$, respectively, as illustrated in Fig-

ure 4.5. Using these factorized small matrices, DE P-Rank calculates the block matrix $\begin{pmatrix} \mathbf{\Sigma}_{11} & \mathbf{\Sigma}_{12} \\ \mathbf{\Sigma}_{21} & \mathbf{\Sigma}_{22} \end{pmatrix}$ and two auxiliary vectors $\mathbf{v}_1, \mathbf{v}_2$, and it returns the similarity matrix $\mathbf{S}$ as the final P-Rank result with no extra approximation errors, due to the choice of low rank $\upsilon = \texttt{Rank}(\mathbf{A}) = 1$ when $\mathbf{Q}$ and $\mathbf{P}$ are completely decomposed. □

Example 4.13 (Homogeneous Graph)**.** DE P-Rank can also be applied to homogeneous domains, such as the web graph $\mathcal{G}_3$ in Figure 4.6. Each vertex in $\mathcal{G}_3$ represents a web page, and each edge corresponds to a hyperlink from one page to another. A similar process for DE P-Rank computing similarity scores for vertices of $\mathcal{G}_3$ (setting $C_{\text{in}} = 0.4$, $C_{\text{out}} = 0.6$, $\lambda = 0.5$) is shown in Figure 4.6. No additional approximation errors arise as the low rank $\upsilon = \texttt{Rank}(\mathbf{A}) = 2$ is selected. □

To complete the proof of Theorem 4.10, we next show that (1) the algorithm DE P-Rank correctly estimates the similarity values; and (2) it has the time complexity bound stated in Theorem 4.10.

(1) Correctness. (i) We first verify that the algorithm returns *exactly* the same similarity results as Eq.(4.11) when $\upsilon = \texttt{Rank}(\mathbf{A})$.

Let $\mathbf{Q} = \mathbf{U_Q}\mathbf{\Sigma_Q}\mathbf{V_Q}^T$ and $\mathbf{P} = \mathbf{U_P}\mathbf{\Sigma_P}\mathbf{V_P}^T$ be two reduced singular value decompositions. Using the Kronecker product property [HJ90] that $(\mathbf{A}\cdot\mathbf{B})\otimes(\mathbf{C}\cdot\mathbf{D}) = (\mathbf{A}\otimes\mathbf{C})\cdot(\mathbf{B}\otimes\mathbf{D})$ yields

$$\mathbf{Q}\otimes\mathbf{Q} = \tilde{\mathbf{U}}_\mathbf{Q}\cdot\tilde{\mathbf{\Sigma}}_\mathbf{Q}\cdot\tilde{\mathbf{V}}_\mathbf{Q}^T, \quad \mathbf{P}\otimes\mathbf{P} = \tilde{\mathbf{U}}_\mathbf{P}\cdot\tilde{\mathbf{\Sigma}}_\mathbf{P}\cdot\tilde{\mathbf{V}}_\mathbf{P}^T,$$

where

$$\tilde{\mathbf{U}}_\mathbf{Q} = \mathbf{U_Q}\otimes\mathbf{U_Q},\ \tilde{\mathbf{\Sigma}}_\mathbf{Q} = \mathbf{\Sigma_Q}\otimes\mathbf{\Sigma_Q},\ \tilde{\mathbf{V}}_\mathbf{Q} = \mathbf{V_Q}\otimes\mathbf{V_Q},$$
$$\tilde{\mathbf{U}}_\mathbf{P} = \mathbf{U_P}\otimes\mathbf{U_P},\ \tilde{\mathbf{\Sigma}}_\mathbf{P} = \mathbf{\Sigma_P}\otimes\mathbf{\Sigma_P},\ \tilde{\mathbf{V}}_\mathbf{P} = \mathbf{V_P}\otimes\mathbf{V_P}.$$

Substituting these into Eq.(4.11) and combining Lemma 4.9 produces

$$\text{vec}\,(\mathbf{S}) = \left(\mathbf{I}_{n^2} - \lambda C_{\text{in}}\,(\mathbf{Q}\otimes\mathbf{Q}) - (1-\lambda)\,C_{\text{out}}\,(\mathbf{P}\otimes\mathbf{P})\right)^{-1}\cdot\text{vec}\,(\mathbf{I}_n)$$

$$= \underbrace{\left(\mathbf{I}_{n^2} - \lambda C_{\text{in}} \tilde{\mathbf{U}}_{\mathbf{Q}} \tilde{\mathbf{\Sigma}}_{\mathbf{Q}} \tilde{\mathbf{V}}_{\mathbf{Q}}^T - (1-\lambda)\, C_{\text{out}} \tilde{\mathbf{U}}_{\mathbf{P}} \tilde{\mathbf{\Sigma}}_{\mathbf{P}} \tilde{\mathbf{V}}_{\mathbf{P}}^T\right)^{-1}}_{=\{\text{using Lemma 4.9}\}=(\mathbf{I}_{n^2}+(\,\tilde{\mathbf{U}}_{\mathbf{Q}}\ \tilde{\mathbf{U}}_{\mathbf{P}}\,)\mathbf{\Sigma}(\,\tilde{\mathbf{V}}_{\mathbf{Q}}\ \tilde{\mathbf{V}}_{\mathbf{P}}\,)^T)\text{vec}(\mathbf{I}_n)} \cdot \text{vec}\left(\mathbf{I}_n\right)$$
$$= \text{vec}\left(\mathbf{I}_n\right) + \begin{pmatrix} \tilde{\mathbf{U}}_{\mathbf{Q}} & \tilde{\mathbf{U}}_{\mathbf{P}} \end{pmatrix} \mathbf{\Sigma} \begin{pmatrix} \tilde{\mathbf{V}}_{\mathbf{Q}} & \tilde{\mathbf{V}}_{\mathbf{P}} \end{pmatrix}^T \text{vec}\left(\mathbf{I}_n\right),$$

where

$$\mathbf{\Sigma} = \begin{pmatrix} \frac{1}{\lambda C_{\text{in}}} \tilde{\mathbf{\Sigma}}_{\mathbf{Q}}^{-1} - \tilde{\mathbf{V}}_{\mathbf{Q}}^T \tilde{\mathbf{U}}_{\mathbf{Q}} & -\tilde{\mathbf{V}}_{\mathbf{Q}}^T \tilde{\mathbf{U}}_{\mathbf{P}} \\ -\tilde{\mathbf{V}}_{\mathbf{P}}^T \tilde{\mathbf{U}}_{\mathbf{Q}} & \frac{1}{(1-\lambda) C_{\text{out}}} \tilde{\mathbf{\Sigma}}_{\mathbf{P}}^{-1} - \tilde{\mathbf{V}}_{\mathbf{P}}^T \tilde{\mathbf{U}}_{\mathbf{P}} \end{pmatrix}^{-1}.$$

(ii) We next verify that in the case of $\upsilon < \texttt{Rank}\left(\mathbf{A}\right)$, the algorithm returns the low-rank υ *approximate* P-Rank similarity with an error bound of ϵ_υ stated in Theorem 4.10.

To simplify the notations, let

$$\mathbf{M}_\upsilon = \mathbf{I}_{n^2} - \lambda C_{\text{in}} \left(\mathbf{Q}_\upsilon \otimes \mathbf{Q}_\upsilon\right) - (1-\lambda)\, C_{\text{out}} \left(\mathbf{P}_\upsilon \otimes \mathbf{P}_\upsilon\right), \tag{4.19}$$

where $\mathbf{Q}_\upsilon$ and $\mathbf{P}_\upsilon$ are the rank υ approximation of $\mathbf{Q}$ and $\mathbf{P}$) respectively, as depicted in Figure 4.4. Recall that the exact closed-form solution of P-Rank similarity Eq.(4.11), which can be equivalently rewritten as

$$\mathbf{M} \cdot \text{vec}\left(\mathbf{S}\right) = (1 - \lambda C_{\text{in}} - (1-\lambda) C_{\text{out}}) \cdot \text{vec}\left(\mathbf{I}_n\right),$$

but due to the low-rank υ approximation, one actually solve

$$\mathbf{M}_\upsilon \cdot \text{vec}(\hat{\mathbf{S}}_\upsilon) = (1 - \lambda C_{\text{in}} - (1-\lambda) C_{\text{out}}) \cdot \text{vec}(\mathbf{I}_n).$$

To estimate the error ϵ_υ, combining the above two equations yields

$$\mathbf{M} \cdot (\text{vec}(\mathbf{S}) - \text{vec}(\hat{\mathbf{S}}_\upsilon)) = (\mathbf{M}_\upsilon - \mathbf{M}) \cdot \text{vec}(\hat{\mathbf{S}}_\upsilon).$$

Let

$$\epsilon_\upsilon = \frac{\|\mathbf{S} - \hat{\mathbf{S}}_\upsilon\|_{\max}}{\|\hat{\mathbf{S}}_\upsilon\|_{\max}} = \frac{\|\text{vec}(\mathbf{S} - \hat{\mathbf{S}}_\upsilon)\|_\infty}{\|\text{vec}(\hat{\mathbf{S}}_\upsilon)\|_\infty}.$$

Since $\mathbf{M}$ was proved in Subsection 4.4.1 to be invertible, pre-multiplying by $\mathbf{M}^{-1}$ and taking ∞-norm on both sides of the above equation produces

$$\epsilon_\upsilon \leq \|\mathbf{M}^{-1}\|_\infty \cdot \|\mathbf{M}_\upsilon - \mathbf{M}\|_\infty.$$

According to Lemma 4.7, we have

$$\|\mathbf{M}^{-1}\|_\infty \leq 1/(1-\lambda C_{\text{in}} - (1-\lambda)\, C_{\text{out}}). \tag{4.20}$$

By the equivalence of norms, it follows that

$$\|\mathbf{M}_\upsilon - \mathbf{M}\|_\infty \leq r\|\mathbf{M}_\upsilon - \mathbf{M}\|_2.$$

We are now ready to find the bound of $\|\mathbf{M}_\upsilon - \mathbf{M}\|_2$.

Lemma 4.14. Let $\mathbf{M}$ and $\mathbf{M}_\upsilon$ be the matrices defined by Eqs.(4.10) and (4.19). Then the following equality holds:

$$\|\mathbf{M}_\upsilon - \mathbf{M}\|_2 \leq \lambda C_{\text{in}}\sigma_1\sigma_{\upsilon+1} + (1-\lambda)\, C_{\text{out}}\bar{\sigma}_1\bar{\sigma}_{\upsilon+1}, \tag{4.21}$$

where σ_i and $\bar{\sigma}_i$ $(i = 1, \upsilon+1)$ are the i-th largest singular values of $\mathbf{Q}$ and $\mathbf{P}$, respectively.

□

Proof. Subtracting Eq.(4.19) from Eq.(4.10) and taking 2-norms of both sides yield

$$\|\mathbf{M}_\upsilon - \mathbf{M}\|_2 \leq \lambda \cdot C_{\text{in}}\|\mathbf{Q}\otimes\mathbf{Q} - \mathbf{Q}_\upsilon\otimes\mathbf{Q}_\upsilon\|_2 + (1-\lambda)\cdot C_{\text{out}}\|\mathbf{P}\otimes\mathbf{P} - \mathbf{P}_\upsilon\otimes\mathbf{P}_\upsilon\|_2.$$

To find an upper bound for $\|\mathbf{Q}\otimes\mathbf{Q} - \mathbf{Q}_\upsilon\otimes\mathbf{Q}_\upsilon\|_2$, let $\mathbf{Q} = \mathbf{U}_\mathbf{Q}\mathbf{\Sigma}_\mathbf{Q}\mathbf{V}_\mathbf{Q}^T$ be a truncated SVD with

$$\mathbf{\Sigma}_\mathbf{Q} = diag\,(\sigma_1, \sigma_2, \cdots, \sigma_r)\,,$$

and $\mathbf{Q}_\upsilon = \mathbf{U}_\mathbf{Q}\mathbf{\Sigma}_{\mathbf{Q}_\upsilon}\mathbf{V}_\mathbf{Q}^T$ be a truncated SVD with

$$\mathbf{\Sigma}_{\mathbf{Q}_\upsilon} = diag(\sigma_1, \cdots, \sigma_\upsilon, \underbrace{0, \cdots, 0}_{r-\upsilon}).$$

Then it follows that

$$\begin{aligned}
&\mathbf{\Sigma}_\mathbf{Q}\otimes\mathbf{\Sigma}_\mathbf{Q} - \mathbf{\Sigma}_{\mathbf{Q}_\upsilon}\otimes\mathbf{\Sigma}_{\mathbf{Q}_\upsilon} \\
&= diag\,(\overbrace{0, \cdots, 0}^{\upsilon}, \sigma_1\sigma_{\upsilon+1}, \cdots, \sigma_1\sigma_r, \\
&\cdots,
\end{aligned}$$

$$\underbrace{0,\cdots,0}_{v},\sigma_v\sigma_{v+1},\cdots,\sigma_v\sigma_r,$$
$$\sigma_{v+1}\sigma_1,\sigma_{v+1}\sigma_2,\cdots,\sigma_{v+1}\sigma_r,$$
$$\cdots,$$
$$\sigma_r\sigma_1,\sigma_r\sigma_2,\cdots,\sigma_r\sigma_r).$$

Since $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_r$ are the nonzero singular values of $\mathbf{Q}$, arranged in non-increasing order, we know by SVD property that

$$\|\mathbf{\Sigma_Q} \otimes \mathbf{\Sigma_Q} - \mathbf{\Sigma_{Q_v}} \otimes \mathbf{\Sigma_{Q_v}}\|_2 = \sigma_1\sigma_{v+1}. \tag{4.22}$$

From the truncated SVDs of $\mathbf{Q}$ and $\mathbf{Q}_v$, it follows that

$$\mathbf{Q}\otimes\mathbf{Q} - \mathbf{Q}_v\otimes\mathbf{Q}_v = (\mathbf{U_Q}\otimes\mathbf{U_Q})\,(\mathbf{\Sigma_Q}\otimes\mathbf{\Sigma_Q} - \mathbf{\Sigma_{Q_v}}\otimes\mathbf{\Sigma_{Q_v}})\,(\mathbf{V_Q}\otimes\mathbf{V_Q})^T.$$

Due to the property that the Kronecker product of two orthogonal matrices is orthogonal [HJ90], it follows that $\mathbf{U_Q}\otimes\mathbf{U_Q}$ and $\mathbf{V_Q}\otimes\mathbf{V_Q}$ are orthogonal. Hence, by SVD and Eq.(4.22) we have

$$\|\mathbf{Q}\otimes\mathbf{Q} - \mathbf{Q}_v\otimes\mathbf{Q}_v\|_2 = \|\mathbf{\Sigma_Q}\otimes\mathbf{\Sigma_Q} - \mathbf{\Sigma_{Q_v}}\otimes\mathbf{\Sigma_{Q_v}}\|_2 = \sigma_1\sigma_{v+1}. \tag{4.23}$$

Similarly, we can obtain

$$\|\mathbf{P}\otimes\mathbf{P} - \mathbf{P}_v\otimes\mathbf{P}_v\|_2 = \bar{\sigma}_1\bar{\sigma}_{v+1} \tag{4.24}$$

with $\bar{\sigma}_i \quad (i = 1, v+1)$ being the i-th largest singular values of $\mathbf{P}$. Substituting Eqs.(4.23) and (4.24) into Eq.(4.21) yields

$$\begin{aligned}\|\mathbf{M}_v - \mathbf{M}\|_2 &\leq \lambda\cdot C_{\text{in}}\cdot\overbrace{\|\mathbf{Q}\otimes\mathbf{Q} - \mathbf{Q}_v\otimes\mathbf{Q}_v\|_2}^{=\sigma_1\sigma_{v+1}} + (1-\lambda)\cdot C_{\text{out}}\cdot\overbrace{\|\mathbf{P}\otimes\mathbf{P} - \mathbf{P}_v\otimes\mathbf{P}_v\|_2}^{=\bar{\sigma}_1\bar{\sigma}_{v+1}}\\ &= \lambda C_{\text{in}}\sigma_1\sigma_{v+1} + (1-\lambda)\,C_{\text{out}}\bar{\sigma}_1\bar{\sigma}_{v+1},\end{aligned}$$

which completes the proof. □

Based on Lemma 4.14, combining Eqs.(4.20) and (4.21) yields

$$\epsilon_v \leq \frac{\lambda C_{\text{in}}\sigma_1\sigma_{v+1}r + (1-\lambda)\,C_{\text{out}}\bar{\sigma}_1\bar{\sigma}_{v+1}r}{1-\lambda C_{\text{in}} - (1-\lambda)\,C_{\text{out}}}.$$

(2) Complexity. We next show that the algorithm DE P-Rank is in $O\left(v^4n^2 + v^2n\right)$ time.

(i) For pre-processing (lines 1-4), (a) it takes $O\left(n^2\right)$ time to compute $\mathbf{Q}$ and $\mathbf{P}$ by normalizing $\mathbf{A}$ (lines 1 and 2); (b) calculating `Rank` () takes $O\left(n^2\right)$ (line 3); (c) `RSVD` () is computed in $O\left(vn^2 + v^2n\right)$ time (line 4). In particular, the diagonal matrices $\mathbf{\Sigma_Q}$ and $\mathbf{\Sigma_P}$ can be stored in two v-dimensional vectors with the entries σ_i and $\bar{\sigma}_i$ $(i = 1, \cdots, v)$ sorted in descending order, respectively. The total cost in this phase is thus $O\left(vn^2 + v^2n\right)$.

(ii) For the similarity computation phase (lines 5-7), we analyze the time complexity as follows. (a) The self-Kronecker product for each matrix requires at most $O\left(v^2n^2\right)$ (line 5). Moreover, since $\mathbf{\Sigma_Q}$ and $\mathbf{\Sigma_P}$ are diagonal, $\tilde{\mathbf{\Sigma}}_{\mathbf{Q}}^{-1}$ and $\tilde{\mathbf{\Sigma}}_{\mathbf{P}}^{-1}$ can be generated in $O\left(v^2\right)$ time. (b) Computing all the submatrices $\mathbf{\Sigma}_{i,j}$ $(i, j = 1, 2)$ from the matrix $\mathbf{\Sigma}$ takes $O\left(v^4n^2\right)$ time (line 6), particularly involving 4 matrix multiplications and 2 diagonal matrix inversions altogether. (c) $\mathbf{s}$ can be computed at most in $O\left(v^2n^2 + v^6\right)$ time (line 7). One can verify that calculating vectors $\mathbf{v}_1, \mathbf{v}_2$ and $\mathbf{s}$ takes $O\left(v^2n\right)$, $O\left(v^6\right)$ and $O\left(v^2n^2\right)$ worst-case time, respectively.

(iii) The last phase (lines 9-10) is in $O\left(n\right)$ time.

Taking (i), (ii) and (iii) together, the algorithm DE P-Rank is in $O\left(v^4n^2 + v^2n\right)$ total time.

4.5.2 P-Rank on Undirected Graphs

After low-rank techniques are devised for P-Rank computation on digraphs, optimization methods in this subsection allow further reducing the computation of P-Rank on undirected graphs.

The key idea behind the optimization is to diagonalize the adjacency matrix $\mathbf{A}$ into $\mathbf{\Lambda}$ and utilize $\mathbf{\Lambda}$ for computing the similarity $\mathbf{S}$ based on a power series representation of P-Rank solution, *i.e.,* $\mathbf{S} = \sum_{k=0}^{+\infty} f(\mathbf{\Lambda}^k)$. Due to $\mathbf{A}$ symmetry for undirected networks, $\mathbf{A}$

is diagonalizable to $\mathbf{\Lambda}$. As a matrix product is associative, the power of diagonal matrix $\mathbf{\Lambda}$ is much easier to calculate than that of $\mathbf{A}$, and thus improves the efficiency of P-Rank computation on undirected graphs.

The main result in this subsection is the following.

Theorem 4.15. *For undirected graphs, the P-Rank similarity* $\mathbf{S}$ *in Eq.*(4.8) *can be solvable in* $O(rn^2)$ *worst-case time, with* r $(\ll n)$ *being the rank of graph adjacency matrix.* □

(A proof will be provided after some discussions.)

Note that SimRank formula over undirected graphs is a special form of P-Rank similarity when $\lambda = 1$ and $C_{\text{in}} = C_{\text{out}}$. In contrast to the $O(n^3 + Kn^2)$-time of SimRank optimization over undirected graphs in our early work [YLL10], Theorem 4.15 further optimizes P-Rank computational time in $O(rn^2)$ $(r \ll n)$ with no need for extra iterations.

One challenging problem underlying the optimization is to characterize P-Rank similarity $\mathbf{S}$ as the eigenvectors of the adjacency matrix over undirected graphs. To address this issue, we propose the following theorem.

Theorem 4.16. *For an undirected graph* $\mathcal{G}$*, let* $\mathbf{A} = (a_{i,j}) \in \mathbb{R}^{n\times n}$ *be the adjacency matrix, and* $\mathbf{D}$ *a diagonal matrix* [8]

$$\mathbf{D} = diag((\sum_{j=1}^{n} a_{1,j})^{-1}, \cdots, (\sum_{j=1}^{n} a_{n,j})^{-1}) \in \mathbb{R}^{n\times n}. \tag{4.25}$$

Then, the P-Rank similarity matrix $\mathbf{S}'$ *in Eq.*(4.9) *can be explicitly represented as* [9]

$$\mathbf{S}' = \mathbf{D}^{1/2}\mathbf{U} \cdot \mathbf{\Psi} \cdot \mathbf{U}^T\mathbf{D}^{1/2}, \tag{4.26}$$

where $\mathbf{\Psi} = (\Psi_{i,j})_{r\times r}$ *with the* (i,j)*-entry being*

$$\Psi_{i,j} = \frac{[\mathbf{U}^T\mathbf{D}^{-1}\mathbf{U}]_{i,j}}{1 - (\lambda \cdot C_{in} + (1-\lambda)\cdot C_{out})\,\Lambda_{i,i}\Lambda_{j,j}}, \tag{4.27}$$

[8] We define $\frac{1}{0} \triangleq 0$, thereby avoiding division by zero when the column/row sum of $\mathbf{A}$ equals 0.
[9] $\mathbf{D}^{1/2}$ is a diagonal matrix whose diagonal entries are the principal square root of those of $\mathbf{D}$.

and $\mathbf{U}$ *and* $\mathbf{\Lambda}$ *are the eigen-decomposition of* $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$ *such that* $\mathbf{U} \in \mathbb{R}^{n\times r}$ *is an orthogonal matrix with its columns being all eigenvectors of* $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$*, and* $\mathbf{\Lambda} = (\Lambda_{i,j}) \in \mathbb{R}^{r\times r}$ *is an diagonal matrix with its diagonal entries being all eigenvalues of* $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$*; and* $[\mathbf{U}^T\mathbf{D}^{-1}\mathbf{U}]_{i,j}$ *denotes the* (i,j)*-entry of* $\mathbf{U}^T\mathbf{D}^{-1}\mathbf{U}$*.* □

Proof. (i) We first give a power series form of $\mathbf{S}'$ in Eq.(4.9). Since the network $\mathcal{G}$ is undirected, its adjacency matrix is symmetric. Therefore, we can easily verify

$$\mathbf{Q} = \mathbf{P} = \mathbf{D}\cdot\mathbf{A}. \tag{4.28}$$

Using Eq.(4.28) to simplify Eq.(4.9) yields

$$\mathbf{S}' = (\lambda C_{\text{in}} + (1-\lambda)\, C_{\text{out}})\,(\mathbf{D}\mathbf{A})\mathbf{S}'(\mathbf{D}\mathbf{A})^T + \mathbf{I}_n. \tag{4.29}$$

The recursive definition of $\mathbf{S}'$ in Eq.(4.29) naturally leads itself to have the following power series representation:

$$\mathbf{S}' = \sum_{k=0}^{+\infty} (\lambda C_{\text{in}} + (1-\lambda)\, C_{\text{out}})^k (\mathbf{D}\mathbf{A})^k ((\mathbf{D}\mathbf{A})^k)^T. \tag{4.30}$$

(ii) We next need to compute $(\mathbf{D}\mathbf{A})^k$ in Eq.(4.30). Observe that $\mathbf{D}\cdot\mathbf{A} = \mathbf{D}^{1/2}\cdot(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2})\cdot\mathbf{D}^{-1/2}$. Then we have

$$\begin{aligned}(\mathbf{D}\cdot\mathbf{A})^k =& \mathbf{D}^{1/2}(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2})\overbrace{\mathbf{D}^{-1/2}\cdot\mathbf{D}^{1/2}}^{=\mathbf{I}}(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2})\\ &\underbrace{\mathbf{D}^{-1/2}\cdot\mathbf{D}^{1/2}}_{=\mathbf{I}}(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2})\mathbf{D}^{-1/2}\cdots\\ =& \mathbf{D}^{1/2}\cdot(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2})^k\cdot\mathbf{D}^{-1/2}.\end{aligned} \tag{4.31}$$

Since $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$ is symmetric, it can be factorized as $\mathbf{U}\cdot\mathbf{\Lambda}\cdot\mathbf{U}^T$ via eigen-decomposition, where $\mathbf{U}\in\mathbb{R}^{n\times r}$ is an orthogonal matrix whose columns are all eigenvectors of $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$, and $\mathbf{\Lambda} = (\Lambda_{i,j}) \in \mathbb{R}^{r\times r}$ is diagonal with its diagonal entries being all eigenvalues of $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$.

Therefore, it follows from $\mathbf{U}^T\cdot\mathbf{U} = \mathbf{I}$ that

$$(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2})^k = (\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T)^k = \mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^T. \tag{4.32}$$

Substituting Eq.(4.32) back into Eq.(4.31) yields

$$(\mathbf{D}\cdot\mathbf{A})^k = \mathbf{D}^{1/2}\cdot\mathbf{U}\mathbf{\Lambda}^k\mathbf{U}^T\cdot\mathbf{D}^{-1/2}. \tag{4.33}$$

(iii) We finally provide a concise expression for the matrix series form of $\mathbf{S}'$ in Eq.(4.30).

To simplify the equation, let

$$C \triangleq \lambda\cdot C_{\text{in}} + (1-\lambda)\cdot C_{\text{out}},$$

$$\mathbf{\Gamma} = (\Gamma_{i,j})_{r\times r} \triangleq \mathbf{U}^T\mathbf{D}^{-1}\mathbf{U}.$$

Applying Eq.(4.33) to Eq.(4.30) produces

$$\begin{aligned}\mathbf{S}' &= \sum_{k=0}^{+\infty} C^k\cdot\mathbf{D}^{\frac{1}{2}}\mathbf{U}\cdot\mathbf{\Lambda}^k\cdot\mathbf{U}^T\mathbf{D}^{-\frac{1}{2}}\cdot\left(\mathbf{D}^{\frac{1}{2}}\mathbf{U}\cdot\mathbf{\Lambda}^k\cdot\mathbf{U}^T\mathbf{D}^{-\frac{1}{2}}\right)^T\\ &= \mathbf{D}^{\frac{1}{2}}\mathbf{U}\cdot\left(\sum_{k=0}^{+\infty} C^k\cdot\mathbf{\Lambda}^k\cdot\mathbf{\Gamma}\cdot\mathbf{\Lambda}^k\right)\cdot\mathbf{U}^T\mathbf{D}^{\frac{1}{2}}\\ &= \mathbf{D}^{1/2}\mathbf{U}\cdot\mathbf{\Psi}\cdot\mathbf{U}^T\mathbf{D}^{1/2},\end{aligned}$$

where

$$\mathbf{\Psi} = \sum_{k=0}^{+\infty} C^k\cdot\begin{pmatrix}(\Lambda_{1,1}\Lambda_{1,1})^k\Gamma_{1,1} & \cdots & (\Lambda_{1,1}\Lambda_{r,r})^k\Gamma_{1,r}\\ \vdots & \ddots & \vdots\\ (\Lambda_{r,r}\Lambda_{1,1})^k\Gamma_{r,1} & \cdots & (\Lambda_{r,r}\Lambda_{r,r})^k\Gamma_{r,r}\end{pmatrix} = \begin{pmatrix}\frac{\Gamma_{1,1}}{1-C\Lambda_{1,1}\Lambda_{1,1}} & \cdots & \frac{\Gamma_{1,r}}{1-C\Lambda_{1,1}\Lambda_{r,r}}\\ \vdots & \ddots & \vdots\\ \frac{\Gamma_{r,1}}{1-C\Lambda_{r,r}\Lambda_{1,1}} & \cdots & \frac{\Gamma_{r,r}}{1-C\Lambda_{r,r}\Lambda_{r,r}}\end{pmatrix}.$$

□

Based on Theorem 4.16, we next prove Theorem 4.15 by providing an algorithm for P-Rank computation over undirected networks.

Algorithm. The algorithm, referred to as UN P-Rank , is shown in Algorithm 4.2. It takes as input a labeled undirected network $\mathcal{G}$, a weighting factor λ, and in- and out-link damping factors C_{in} and C_{out}; and it returns the P-Rank similarity matrix $\mathbf{S} = (s_{i,j})_{n\times n}$ of all vertex-pairs in $\mathcal{G}$.

The algorithm UN P-Rank works as follows. It first initializes the adjacency matrix $\mathbf{A}$ of the network $\mathcal{G}$ (line 1). Utilizing $\mathbf{A}$, it then computes the auxiliary diagonal matrix

Algorithm 4.2: UN P-Rank $(\mathcal{G}, \lambda, C_{\text{in}}, C_{\text{out}})$

Input : undirected web graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$,

damping factors C_{in} and C_{out}, weight factor λ.

Output: similarity matrix $\mathbf{S}$.

1 initialize the adjacency matrix $\mathbf{A}$ of $\mathcal{G}$.

2 compute the diagonal matrix $\mathbf{D} = diag(d_{1,1}, \cdots, d_{n,n})$:

for $i \leftarrow 1, 2, \cdots, n$ **do**

  if $\sum_{j=1}^{n} a_{i,j} \neq 0$ **then** $d_{i,i} \leftarrow (\sum_{j=1}^{n} a_{i,j})^{-1}$

  else $d_{i,i} \leftarrow 0$.

3 compute the auxiliary matrix $\mathbf{T} \leftarrow \mathbf{D}^{1/2} \cdot \mathbf{A} \cdot \mathbf{D}^{1/2}$.

4 decompose $\mathbf{T}$ into the diagonal $\mathbf{\Lambda} = diag(\Lambda_{1,1}, \cdots, \Lambda_{r,r})$ and the orthogonal $\mathbf{U}$ *s.t.* $\mathbf{T} = \mathbf{U} \cdot \mathbf{\Lambda} \cdot \mathbf{U}^T$.

5 compute the auxiliary matrix $\mathbf{\Gamma} = (\Gamma_{i,j})_{n \times n} \leftarrow \mathbf{U}^T \cdot \mathbf{D}^{-1} \cdot \mathbf{U}$ and $\mathbf{V} \leftarrow \mathbf{D}^{1/2} \cdot \mathbf{U}$ and the scalar $C \leftarrow \lambda C_{\text{in}} + (1 - \lambda) C_{\text{out}}$.

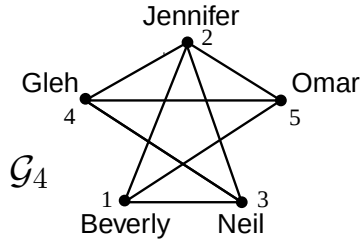
6 compute the matrix $\mathbf{\Psi} = (\psi_{i,j})_{n \times n}$ whose entry $\psi_{i,j} \leftarrow \Gamma_{i,j} / (1 - C \cdot \Lambda_{i,i} \cdot \Lambda_{j,j})$.

7 compute the P-Rank similarity matrix $\mathbf{S} \leftarrow (1 - C) \cdot \mathbf{V} \cdot \mathbf{\Psi} \cdot \mathbf{V}^T$.

8 **return** $\mathbf{S}$.

$\mathbf{D}$ whose (i, i)-entry equals the reciprocal of the i-th column sum of $\mathbf{A}$ if this reciprocal exists, and 0 otherwise (line 2). UN P-Rank then uses QR eigen-decomposition [GL96] to factorize $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$ as $\mathbf{U}\mathbf{\Lambda}\mathbf{U}^T$, in which all columns of $\mathbf{U}$ are the eigenvectors of $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$, and all diagonal entries of $\mathbf{\Lambda}$ are the corresponding eigenvalues of $\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$ (lines 3-4). Utilizing $\mathbf{U}$ and $\mathbf{\Lambda}$, it calculates $\mathbf{\Psi}$ (lines 5-6) to obtain the similarity matrix $\mathbf{S}$ (lines 7-8), which can be justified by Eqs.(4.26) and (4.27).

Example 4.17. Consider an undirected friendship graph $\mathcal{G}_4$. Each vertex corresponds to a person, and there is an edge between two people whenever they are friends. The detailed process of computing $\mathbf{S}$ is depicted in Figure 4.7 step by step without the need for extra iterations. UN P-Rank returns $\mathbf{S}$ as the final similarity result, which is exactly



$$\mathbf{A} = \begin{pmatrix} 0&1&1&0&1\\ 1&0&1&1&1\\ 1&1&0&1&0\\ 0&1&1&0&1\\ 1&1&0&1&0 \end{pmatrix} \xrightarrow{①} \mathbf{D} = \{\text{Using Eq.(26)}\} = \begin{pmatrix} .333&&&&0\\ &.25&&&\\ &&.333&&\\ &&&.333&\\ 0&&&&.333 \end{pmatrix}$$

$$\xrightarrow{④} \mathbf{\Lambda} = \text{eigval}(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}) = \begin{pmatrix} -.667&&0\\ &-.333&\\ 0&&1 \end{pmatrix}$$

$$\mathbf{U} = \text{eigvec}(\mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}) = \begin{pmatrix} -.5&-.25&.433\\ 0&.866&.5\\ .5&-.25&.433\\ -.5&-.25&.433\\ .5&-.25&.433 \end{pmatrix} \xrightarrow{⑤} \mathbf{\Gamma} = \mathbf{U}^T\mathbf{D}^{-1}\mathbf{U} = \begin{pmatrix} 3&0&0\\ 0&3.75&.433\\ 0&.433&3.25 \end{pmatrix}$$

$$\mathbf{\Psi} = \{\text{Using Eq.(28)}\} = \begin{pmatrix} 4.656&0&0\\ 0&4.116&.342\\ 0&.342&16.25 \end{pmatrix} \xrightarrow{⑥} \mathbf{S} = \{\text{Using Eq.(27)}\} = \begin{pmatrix} .293&.157&.138&.293&.138\\ .157&.372&.157&.157&.157\\ .138&.157&.293&.138&.293\\ .293&.157&.138&.293&.138\\ .138&.157&.293&.138&.293 \end{pmatrix} \xrightarrow{⑦}$$

Figure 4.7: How UN P-Rank works on undirected $\mathcal{G}_4$

the solution to Eq.(4.8). □

To complete the proof of Theorem 4.15, we next show that algorithm UN P-Rank requires quadratic time in the number of vertices.

Complexity. (i) In lines 2-3, computing the diagonal $\mathbf{D}$ and $\mathbf{T} = \mathbf{D}^{1/2}\mathbf{A}\mathbf{D}^{1/2}$ needs $O(m)$ and $O(n^2)$ time, respectively, with n and m being the number of vertices and edges in $\mathcal{G}$ respectively. (ii) In line 4, QR factorization of $\mathbf{T}$ into the orthogonal $\mathbf{U}$ and the diagonal $\mathbf{\Lambda}$ requires $O(rn^2)$ worse-case time, with r ($\ll n$) being the rank of $\mathbf{A}$. (iii) In lines 5-7, computing the auxiliary matrices $\mathbf{\Gamma}, \mathbf{V}, \mathbf{\Psi}$ and the similarity matrix $\mathbf{S}$ yields $O(r^3), O(rn), O(r^2)$ and $O(r^2n + rn^2)$, respectively, which can be further bounded by $O(rn^2)$. Combining (i), (ii) and (iii), the total time of UN P-Rank is in $O(rn^2)$ ($r \ll n$).

4.6 Experimental Evaluation

In this section, a comprehensive empirical study of our P-Rank methods is presented. By using real and synthetic data, four sets of experiments are conducted to evaluate the effectiveness and efficiency of our approaches.

4.6.1 Experimental Setting

We used real-life data and synthetic data.

(1) Real-life data. The real data were taken from DBLP[10]. We extracted the 10-year (from 1998 to 2007) author-paper information and picked up papers published on 6 major conferences ("ICDE", "VLDB", "SIGMOD", "WWW", "SIGIR" and "KDD"). Every two years made a time step. For each time step, we built a co-authorship network incrementally from the one of previous time step. We chose the relationship that there is an edge between authors if one author wrote a paper with another. The sizes of these DBLPnetworks are as follows:

DBLP	98-99	98-01	98-03	98-05	98-07
m	5,929	13,441	24,762	39,399	54,844
n	1,525	3,208	5,307	7,984	10,682

(2) Synthetic data. We also used a C++ boost generator to produce graphs, controlled by two parameters: the number n of vertices, and the number m of edges. We then produced five web graphs (RANDdata) by varying the vertex size n from 100K to 1M with edges randomly chosen.

(3) Algorithms. We have implemented the following algorithms in C++: (a) our algorithms DE P-Rank and UN P-Rank ; (b) the conventional P-Rank iterative algorithm Naive [ZHS09] with the radius-based pruning technique; (c) the memoization-based algorithm Memo [LVGT10] applied to P-Rank computation; (d) SimRank optimized algorithm AUG [YLL10] over undirected graphs.

[10]http://www.informatik.uni-trier.de/~ley/db/

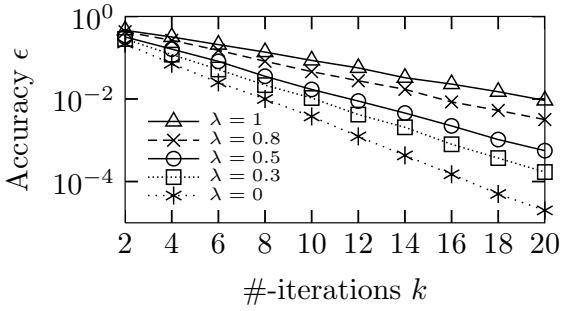


Figure 4.8: ϵ *w.r.t.* k on 1M RAND

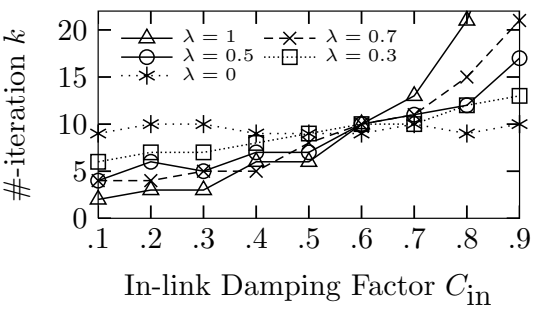


Figure 4.9: k *w.r.t.* C_{in} 1M RAND

The experiments were run on a machine with a Pentium(R) Dual-Core (2.0GHz) CPU and 4GB RAM, using Windows Vista. Each experiment was repeated 5 times and the average is reported.

For fairness of comparison, the following parameters were used as default values (unless otherwise specified).

Notation	Description	Default Value
λ	weighting factor	0.5
C_{in}	in-link damping factor	0.8
C_{out}	out-link damping factor	0.6
υ	low approximation rank	$50\% \times \texttt{Rank}(\mathcal{G})$
ϵ	desired accuracy	0.001

4.6.2 Experimental Results

Exp-1: Accuracy

We first investigate the impacts of weighting factor λ and damping factors C_{in} and C_{out} on P-Rank accuracy, using RANDdata. Here, the accuracy is measured by the absolute difference between the iterative and exact P-Rank [11].

Using RANDwith 1M vertices, we considered various λ from 0 to 1. For each vertex-pair, we varied the number of iterations k from 2 to 20. The results are reported in Figure 4.8, in which the logarithmic scale is chosen across the y-axis to provide an illustrative

[11] To select the P-Rank "exact" solution $s(\cdot,\cdot)$, we used *the Cauchy's criterion for convergence* and regarded the 100th iterative $s^{(100)}(\cdot,\cdot)$ score as the "exact" one *s.t.* $|s^{(100)}(\cdot,\cdot) - s^{(101)}(\cdot,\cdot)| \ll 1 \times 10^{-10}$.

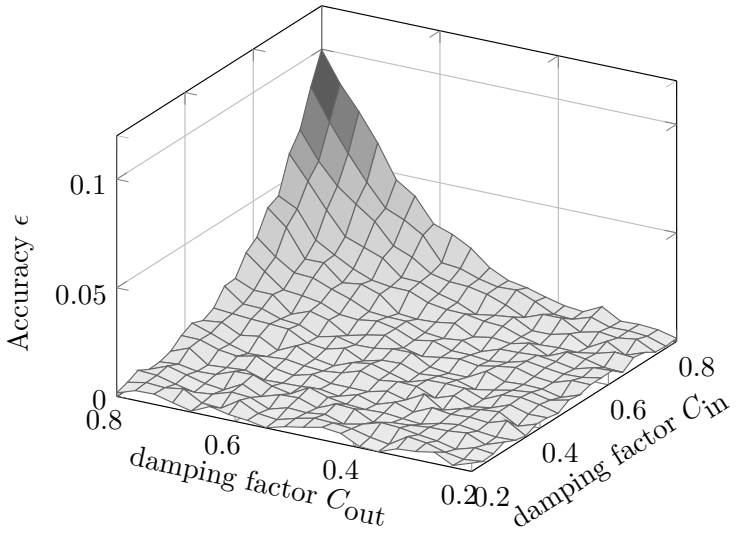


Figure 4.10: ϵ *w.r.t.* C_{in} and C_{out} on Real Data (DBLP)

look for the asymptotic rate of P-Rank convergence. For each fixed λ, the downward lines for P-Rank iterations reveal an exponential accuracy as k increases, as expected in Theorem 4.1. We also observe that the larger λ dramatically increases the slope of a line, which tells that increasing the weighting factor may decrease the convergence rate for P-Rank iteration, as expected.

Figure 4.9 shows the effects of damping factors *w.r.t.* the number of P-Rank iterations required for attaining the fixed accuracy. Fixing $\epsilon = 0.001$ and $C_{\text{out}} = 0.6$, we varied C_{in} from 0.1 to 0.9. (For space constraints, a similar result of varying C_{out} is omitted.) It can be noticed that when $\lambda = 0$, the curve in Figure 4.9 approaches a horizontal line. This is because in this case P-Rank boils down to the reversed SimRank with no in-links considered, which makes C_{in} insensitive to the final P-Rank score. When $0 < \lambda \leq 1$, the iteration number k shows a general increasing tendency as C_{in} grows. This tells that small choices of damping factors may reduce the number of iterations required for a fixed accuracy, and hence, improves the efficiency of P-Rank.

To evaluate the impact of both C_{in} and C_{out} *w.r.t.* the accuracy, we used the real DBLPdata. We only report the result on DBLP1998-2007 data in Figure 4.10, which shows a 3D shaded surface from the average of accuracy value for all vertex-pairs on

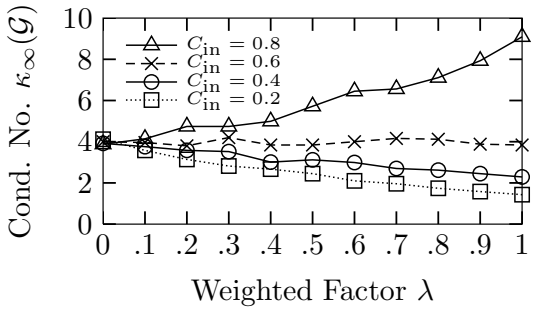


Figure 4.11: κ_∞ *w.r.t.* λ on 1M RAND

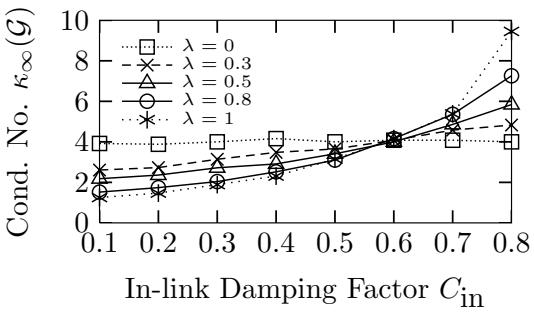


Figure 4.12: κ_∞ *w.r.t.* C_{in} on 1M RAND

z-axis when we fixed $k = 10$ and $\lambda = 0.5$, and varied C_{in} and C_{out} in x-axis and y-axis, respectively. It can be seen that the residual becomes huge only when C_{in} and C_{out} are both increasing to 1; and the iterative P-Rank is accurate when C_{in} and $C_{\text{out}} < 0.6$. This explains why small choices of damping factors are suggested in P-Rank iteration.

Exp-2: Stability

To evaluate P-Rank stability, we investigate the effects of the weighting factors λ and the damping factors C_{in} and C_{out} upon the P-Rank condition number κ_∞.

We fixed $C_{\text{out}} = 0.6$ and varied C_{in} from 0.2 to 0.8. The result over 1M RANDdata is reported in Figure 4.11, in which x-axis denotes λ ranging from 0 to 1. Accordingly, by fixing $C_{\text{out}} = 0.6$ and varying λ from 0 and 1, Figure 4.12 visualizes the impact of C_{in} on P-Rank stability.

Both results in Figures 4.11 and 4.12 show that increasing λ induces a large P-Rank condition number when $C_{\text{in}} > 0.6$. Notice that for different C_{in}, there is one common point $(\lambda, \kappa_\infty) = (0, 4)$ of intersection of all curves in Figure 4.11; correspondingly, in the extreme case of $\lambda = 0$, the curve in Figure 4.12 approaches to a horizontal line. These indicate that varying C_{in} when $\lambda = 0$ has no effect on the stability κ_∞ of P-Rank, for in this case only the contribution of out-links is considered for computing P-Rank. When $C_{\text{in}} < 0.6$, however, $\kappa_\infty(\mathcal{G})$ is decreased as λ grows. This tells that small weighting factor and damping factors yield small P-Rank condition numbers, and thus make the P-Rank well-conditioned, as expected in Theorem 4.10.

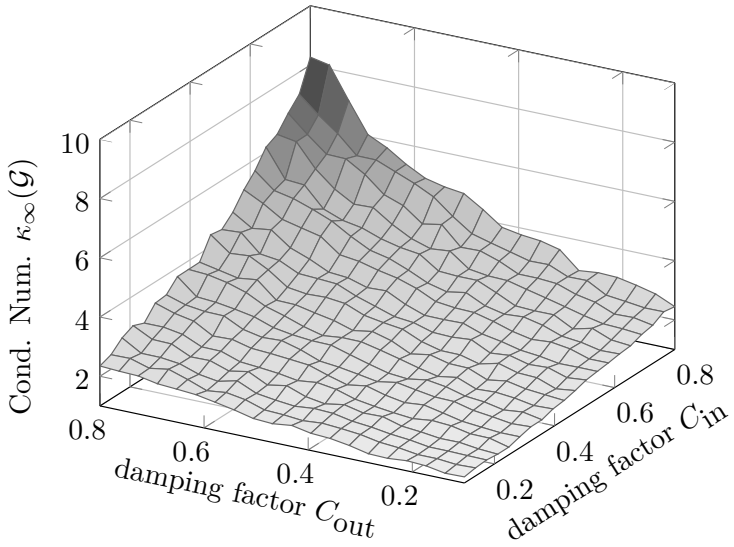


Figure 4.13: κ_∞ *w.r.t.* C_{in} and C_{out} on Real Data (DBLP)

For real-life datasets, Figure 4.13 depicts in 3D view the impacts of both C_{in} and C_{out} on P-Rank stability over 1M DBLPdata, in which x- and y-axis represent in- and out-link damping factors respectively, and z-axis stands for the P-Rank condition number. The result demonstrates that P-Rank is comparatively stable when both C_{in} and C_{out} are small (less than 0.6). However, when C_{in} and C_{out} are approaching to 1, P-Rank becomes ill-conditioned and less useful since small perturbations in similarity computation may cause P-Rank scores drastically altered, which carries the risk of producing nonsensical similarity results. In light of this, small choices of damping factors are preferable.

Exp-3: Time Efficiency

We evaluated the time efficiency of DE P-Rank and UN P-Rank and their scalability using synthetic and real data.

Figure 4.14 compares the running time of DE P-Rank and UN P-Rank with those of Memo and Naive on synthetic (directed and undirected) RANDand real DBLPdata. We use the logarithmic scale on the CPU time (y-axis). The iteration number for Naive and Memo is set to 10. Note that different time unit is chosen across the vertical axis in the two plots of Fig.4.14 to provide a clear look for each bar shape. (i) By varying the

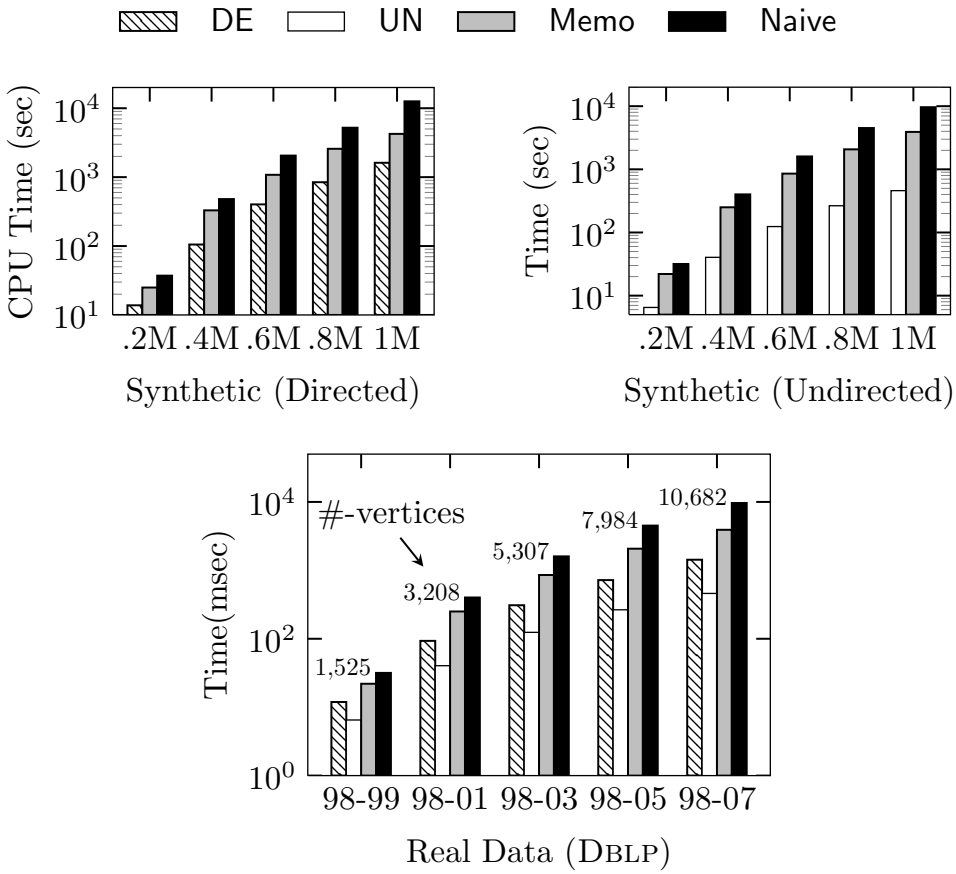


Figure 4.14: Scalability & Computational Time

number of vertices from 200K to 1M, the result on RANDindicates that DE P-Rank and UN P-Rank outperformed Memo and Naive ; the computational time of UN P-Rank has almost 0.5 and 1 order of magnitude faster than Naive , respectively, *i.e.,* utilizing the non-iterative paradigms for P-Rank estimation is highly efficient. In most cases, there are a great number of repeated iterations for Naive and Memo to reach a fixed-point of P-Rank scores, which impedes their scalability in similarity computation. (ii) The result on DBLPdemonstrates the CPU time with respect to the number of nodes for P-Rank estimation when the sizes of DBLPare increased from 1.5K to 10K. In all cases, UN P-Rank performed the best, DE P-Rank the second, by taking advantage of their non-iterative paradigms.

To compare the performances of UN P-Rank and AUG , we applied them to compute SimRank similarities over synthetic RANDdata, by setting $\lambda = 1$ for UN P-Rank (a special

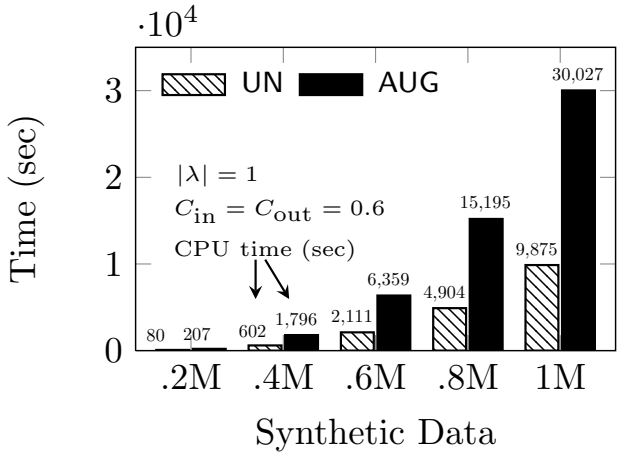


Figure 4.15: UN P-Rank *vs.* AUG on RAND

Rank	DE ($\upsilon/r = 0.5$)	DE ($\upsilon/r = 0.8$)	Naive
1	Shivnath Babu	Shivnath Babu	Shivnath Babu
2	Yingwei Cui	Yingwei Cui	Yingwei Cui
3	Jun Yang	Chris Olston	Chris Olston
4	Chris Olston	Jun Yang	Jun Yang
5	David J. DeWitt	Arvind Arasu	Rajeev Motwani
6	Arvind Arasu	Rajeev Motwani	Arvind Arasu
7	Rajeev Motwani	Anish Das Sarma	Utkarsh Srivastava
8	Utkarsh Srivastava	Alon Y. Halevy	David J. DeWitt
9	Glen Jeh	Omar Benjelloun	Omar Benjelloun
10	Alon Y. Halevy	David J. DeWitt	Alon Y. Halevy

Figure 4.16: Top-10 Similar Authors of "Jennifer Widom" on DBLP

case of P-Rank without out-links consideration). Figure 4.15 reports the result over synthetic RANDdata. It can be seen that UN P-Rank runs approx. 3 times faster than AUG though the CPU time of the UN P-Rank and AUG are of the same order of magnitude. The reason is that after eigen-decomposition, AUG still requires extra iterations to be performed in the small eigen-subspace, which takes a significant amount of time, whereas UN P-Rank can straightforwardly compute similarities in terms of eigenvectors with no need for iterations.

We further evaluate the ground truth calculated by DE P-Rank when varying the approximation ratio υ/r on DBLP(98-07) dataset to retrieve the top-k most similar authors for a given query u. Figure 4.16 depicts the top-10 ranked results for the query

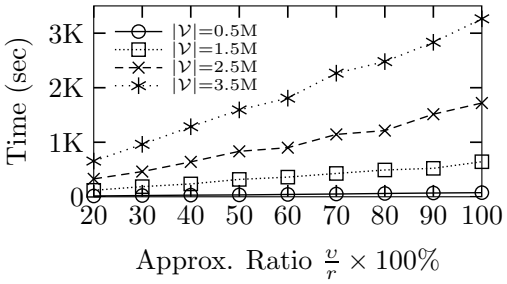


Figure 4.17: Amortized Time on Real Data

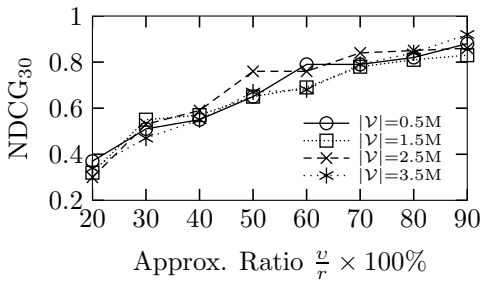


Figure 4.18: Effect of Density

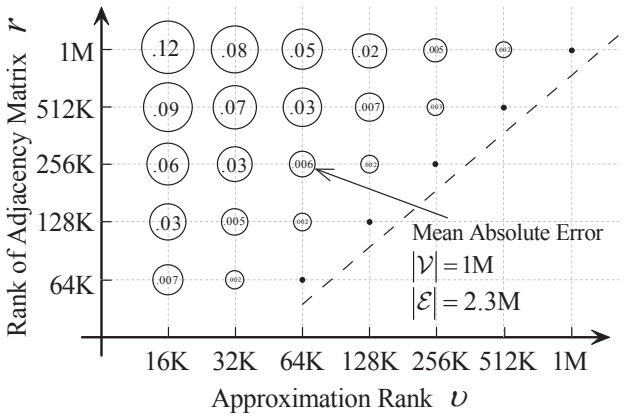


Figure 4.19: ν on Synthetic RAND (1M)

"Jennifer Widom" according to the similarity scores returned by DE P-Rank and Naive , respectively. These members were frequent co-authors of the 6 major conference papers with "Jennifer Widom" from 1998 to 2007. It can be noticed that the ranked results for different algorithms on DBLP(98-07) are practically acceptable and obey our common sense pretty well. When υ/r increases from 0.5 to 0.8, the similarities calculated by DE P-Rank tend to preserve the relative order of Naive . Hence, DE P-Rank can be effectively used for P-Rank top-k nearest neighbor search on real networks.

Exp-4: Effects of υ

For DE P-Rank algorithm, we next investigate the impact of approximation rank υ and graph adjacency matrix rank r on similarity estimation, using synthetic data.

We use 4 graphs with the size $|\mathcal{V}|$ ranging from 0.5M to 3.5M, $2 \times |\mathcal{V}|$ edges, and 128K different vertex attributes. Fixing $|\mathcal{V}|$, we vary υ from $10\% \times r$ to $90\% \times r$. The results are

reported in Figures 4.17 and 4.18, which visualizes the low-rank υ as a speed-accuracy trade-off. When υ becomes increasingly close to r (*i.e.,* the radio $\frac{\upsilon}{r}$ approaches 1), high accuracy (NDCG_{30}) could be expected (Figure 4.18), but more running time needs to be consumed (Figure 4.17). This tells that adding approximation rank υ induces smaller errors for similarity estimation while increasing the computational time, up to a point of r when no extra approximation errors can be reduced.

We also use the following *mean absolute error* (MAE) to evaluate the effects of υ on the DE P-Rank accuracy.

$$\text{MAE} = \frac{1}{n^2} \sum_{u,v \in \mathcal{V}} |s(u,v) - \hat{s}(u,v)|,$$

where $s(u,v)$ is the exact similarity from Naive, and $\hat{s}(u,v)$ the estimate one from DE P-Rank .

Fixing $|\mathcal{V}|$ and $|\mathcal{E}|$, we generate 5 graphs with the rank r growing geometrically from 64K to 1M. Figure 4.19 compares the accuracy (MAE) of estimation when we vary υ for each graph, in which the x-axis represents the approximation rank used, the y-axis gives the rank r of adjacency matrix for a given graph. The number enclosed in a circle $\circ$ corresponds to the mean absolute error incurred by low-rank approximation, and the circles are scaled proportionally to this number. Observe that (1) in all the cases, the circle becomes smaller as υ approaches r, and (2) the sizes of the circles in the diagonal direction ($\nearrow$) remain almost the same, where the ratio $\frac{\upsilon}{r}$ is a constant. This reveals that the choice of approximation ratio $\frac{\upsilon}{r}$ is pivotal to the accuracy of DE P-Rank algorithm.

4.7 Related Work

There has been a surge of studies on link-based analysis (*e.g.,* [YLZ+13b, Kle99, CZDC10, Sma73, ZCY09, XFF+05, JW02, YZL+12, PBMW99]) in recent years. PageRank became popular since the famous result of Page *et al.* [PBMW99] was used by the Google for ranking web pages. Since then, a host of new ranking algorithms for web pages have been developed. The famous results include HITS [Kle99] proposed by Jon *et al.* (now

used in `www.ask.com`), SimRank [AMC08, YLZ13a, FNSO13, JW02], SimFusion [CZDC10, YLZ+12, XFF+05] and P-Rank [ZHS09].

SimRank [JW02] is a recursive structural similarity measure based on the intuition that "*two objects are similar if they are related to similar objects* ", which extends the bibliographic coupling and co-citation [Sma73, Ams72] beyond the local neighborhood to the entire graph. Several optimization problems were studied for SimRank estimation, including iterative amortization-based methods [LVGT10, YLZ13a], probabilistic methods [FR05], matrix-based methods [YLL10, FNSO13], dynamic and parallel methods [LHH+10, HFLC10].

More recently, Zhao *et al.* [ZHS09] presented a new P-Rank model when noticing the limited information problem of SimRank—the similarity scores are only determined by their in-link relationships. P-Rank refines SimRank by jointly considering both in- and out-links of entity pairs. The conventional algorithm for computing P-Rank is based on the fixed-point iteration, yielding $O(kn^4)$ time. To optimize the computational time, a similar memoization approach for SimRank [LVGT10] can be applied to P-Rank, which improves the time to $O(kn^3)$. Zhao *et al.* [ZHS09] also proposed a radius- or category-based pruning technique that can further reduce the computation of P-Rank to $O(kd^2n^2)$. However, this approach is heuristic in nature, and the accuracy of the pruning result is not explainable. In comparison, we focus on the problems of P-Rank accuracy, stability and computational efficiency.

Closer to this work are [YLL10, LVGT10]. For undirected graphs, a time-efficient algorithm AUG for SimRank computation was proposed in [YLL10], which is in $O(n^3 + kn^2)$ time. In contrast, we further improve [YLL10] (i) by providing a non-iterative $O(rn^2)$-time algorithm that can explicitly characterize the similarity solution, and (ii) by extending SimRank to the general P-Rank measure. An accuracy estimation for SimRank was addressed in [LVGT10]. However, by directly port the SimRank bound [LVGT10] to P-Rank, we observed that the simple linear combination of the weighted bound in [LVGT10] (*i.e.*, $\lambda \cdot {C_{\text{in}}}^{k+1} + (1-\lambda) \cdot {C_{\text{out}}}^{k+1}$) is not adaptive to P-Rank since

the out-links of P-Rank also have a recursive impact on the similarity of different pairs of vertices. It is challenging to find a new bound for P-Rank accuracy estimation.

4.8 Conclusions

In this chapter, we have studied the problems of P-Rank assessment on large networks. Firstly, we have proposed an accuracy estimate for the P-Rank iteration, by finding out the exact number of iterations needed to attain a given accuracy. Secondly, we have obtained a tight bound for the P-Rank condition number to analyze the stability of P-Rank to show how the weighting factor and the damping factors affect the stability of P-Rank. Finally, we have also devised efficient algorithms to compute P-Rank similarity in $O(r^4n^2 + r^2n)$ time for digraphs, and $O(rn^2)$ time for undirected graphs. The empirical results on both synthetic and real datasets show that our methods achieve high performance and result quality.

Chapter 5

Incremental Random Walk with Restart

5.1 Introduction

Measuring node proximities in a network is one of the key tasks of web search. Due to various applications in recommender systems and social networks, many proximity metrics have come into play. For instance, Brin and Page [PBMW99] invented PageRank to determine the ranking of web pages. Jeh and Widom [JW02] proposed SimRank to assess node-to-node proximities.

Random Walk with Restart (RWR) [TFP06] is one of such useful proximity metrics for ranking nodes in order of relevance to a query node. In RWR, the proximity of node u *w.r.t.* query node q is defined as the limiting probability that a random surfer, starting from q, and then iteratively either moving to one of its out-neighbors with probability weighted by the edge weights, or restarting from q with probability c, will eventually arrive at node u. Recently, RWR has received increasing attention (*e.g.,* for collaborative filtering [FNOK12] and image labeling [WJZZ06]) since it can fairly capture the global structure of graphs, and relations in interlinked networks [TFP08].

Previous RWR computing methods, however, are based on static networks, which is

costly: Given a graph $G(V, E)$, and a query $q \in V$, k-dash [FNOK12] yields, in the worse case, $O(|V|^2)$ time and space, which, in practice, can be bounded by $O(|E| + |V|)$, to find top-k highest proximity nodes. B_LIN and NB_LIN [TFP06] need $O(|V|^2)$ time and space for computing all node proximities. In general, real graphs are often constantly updated with small changes. This calls for the need for incremental algorithms to compute proximities.

5.1.1 Problem Statement

In this chapter, we study the following problem for incremental RWR assessment:

Problem (Incremental Update for RWR)

Given a graph G, proximities $\mathbf{P}$ for G, changes ΔG to G, a query node q, and a restarting probability $c \in (0, 1)$.

Compute changes to the proximities *w.r.t.* q exactly.

Here, $\mathbf{P}$ is a proximity matrix whose entry $[\mathbf{P}]_{i,j}$ denotes the proximity of node i *w.r.t.* query j, and ΔG is comprised of a set of edges to be inserted into or deleted from G.

In contrast with the existing *batch* algorithms [FNOK12, TFP06] that recompute the updated proximities from scratch, our incremental algorithm can exploit the dynamic nature of graphs by pre-computing proximities only once on the entire graph via a batch algorithm, and then *incrementally* computing their changes in response to updates. The response time of RWR can be greatly improved by maximal use of previous computation, as shown in Example 5.1.

Example 5.1. Figure 5.1 depicts a graph G, taken from [FNOK12]. Given the query u_2, the old proximities $\mathbf{P}$ for G, and $c = 0.2$, we want to compute new proximities *w.r.t.* u_2 when there is an edge (u_1, u_5) inserted into G, denoted by ΔG. The existing methods, k-dash and B_LIN, have to recompute the new proximities in $G \cup \Delta G$ from scratch, without using the previously computed proximities in G, which is costly. However, we observe that the increment $[\Delta\mathbf{P}]_{\star,u_2}$[1] to the old $[\mathbf{P}]_{\star,u_2}$ is the linear combination of $[\mathbf{P}]_{\star,u_1}$ and

[1] $[\mathbf{X}]_{\star,j}$ denotes the j-th column vector of matrix $\mathbf{X}$.

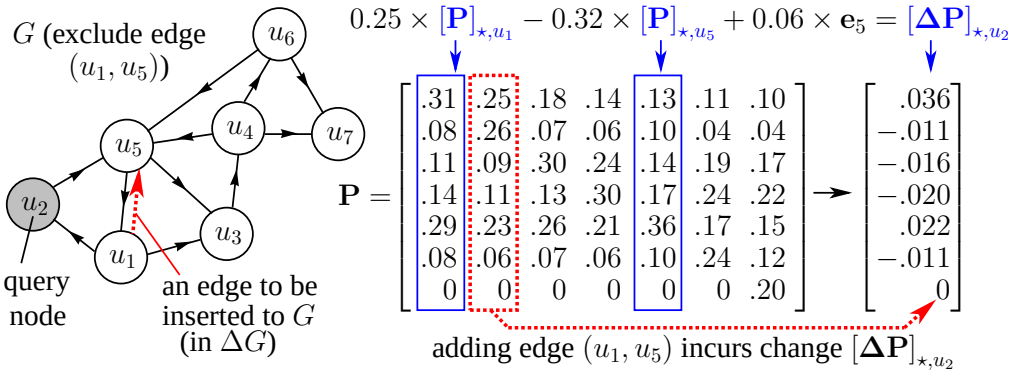


Figure 5.1: Computing RWR Incrementally

$[\mathbf{P}]_{\star,u_5}$, that is,

$$[\mathbf{\Delta P}]_{\star,u_2} = \alpha \cdot [\mathbf{P}]_{\star,u_1} + \beta \cdot [\mathbf{P}]_{\star,u_5} + \lambda \cdot \mathbf{e}_5 \ ^2 \tag{5.1}$$

with $\alpha = 0.25$, $\beta = -0.32$, $\lambda = 0.06$. Hence, there are opportunities to incrementally compute the changes $[\mathbf{\Delta P}]_{\star,u_2}$ by fully utilizing the old proximities of $\mathbf{P}$. As opposed to k-dash and B_LIN involving *matrix-vector multiplications*, computing $[\mathbf{\Delta P}]_{\star,u_2}$ via Eq.(5.1) only needs *vector scaling and additions*, thus greatly improving the response time. □

As suggested by Example 5.1, when the graph G is updated, it is imperative to incrementally compute new proximities by leveraging information from the old proximities. However, it is a grand challenge to characterize the changes $[\mathbf{\Delta P}]_{\star,q}$ in terms of a linear combination of the columns in old $\mathbf{P}$, since it seems hard to determine the scalars α, β, λ for Eq.(5.1). Worse still, much less is known about how to extract a subset of columns from the old $\mathbf{P}$ (*e.g.,* why $[\mathbf{P}]_{\star,u_1}$ and $[\mathbf{P}]_{\star,u_5}$ are chosen from $\mathbf{P}$ in Eq.(5.1)), to express the changes $[\mathbf{\Delta P}]_{\star,q}$.

5.1.2 Chapter Outlines

This chapter aims to tackle these problems. To the best of our knowledge, this makes the first effort to study incremental RWR computing in evolving graphs, with no loss of exactness.

[2] $\mathbf{e}_i$ is the $|V| \times 1$ unit vector with a 1 in the i-th entry.

- We first consider *unit update*, *i.e.,* a single-edge insertion or deletion, and derive an elegant formula that characterizes the proximity changes as a linear combination of the columns from the old proximity matrix.
- We then devise an incremental algorithm for *batch update*, *i.e.,* a list of edge deletions and insertions mixed together, and show that any node proximity can be computed in $O(1)$ time for every edge update, with no sacrifice in accuracy.
- Our empirical study demonstrates that the incremental approach greatly outperforms k-dash [FNOK12], a batch algorithm that is reported as the best for RWR proximity computing, when networks are constantly updated.

The remainder of the chapter is organized as follows. Section 5.2 overviews the background of RWR. The incremental RWR method is investigated in Section 5.3. Section 5.4 presents experimental results, followed by related work in Section 5.5. Section 5.6 concludes this chapter.

5.2 Preliminaries

We formally overview the background of this chapter. Graphs studies here are directed graphs with no multiple edges.

RWR Formula [FNOK12]. In a graph $G = (V, E)$, let $\mathbf{A}$ be the transition matrix (*i.e.,* column normalized adjacency matrix) of G, whose entry $[\mathbf{A}]_{u,v} = \frac{1}{d_v}$ if $(u, v) \in E$, and 0 otherwise. Here, d_v denotes the in-degree of v.

Given query node $q \in V$, and restart probability $c \in (0, 1)$, the proximity of node u *w.r.t.* q, denoted by $[\mathbf{P}]_{u,q}$, is recursively defined as follows:

$$[\mathbf{P}]_{\star,q} = (1 - c) \cdot \mathbf{A} \cdot [\mathbf{P}]_{\star,q} + c \cdot \mathbf{e}_q \tag{5.2}$$

where $[\mathbf{P}]_{\star,q}$ is the $|V| \times 1$ *proximity vector w.r.t.* q (*i.e.,* the q-th column of matrix $\mathbf{P}$), whose u-th entry equals to $[\mathbf{P}]_{u,q}$, and $\mathbf{e}_q$ is the $|V| \times 1$ unit *query vector*, whose q-th entry is 1, and 0 otherwise.

Intuitively, $[\mathbf{P}]_{u,q}$ is the limiting probability, denoting the long-term visit rate of node u, given a bias toward query q.

The RWR proximity defined in Eq.(5.2) can be rewritten as

$$[\mathbf{P}]_{\star,q} = c(\mathbf{I} - (1-c)\cdot\mathbf{A})^{-1}\cdot\mathbf{e}_q \tag{5.3}$$

where $\mathbf{I}$ is the $|V|\times|V|$ identity matrix.

Existing methods of computing RWR are in a batch style, with the aim to accelerate the matrix inversion in Eq.(5.3). For instance, k-dash [FNOK12] uses LU decomposition and an incremental pruning strategy to speed up the matrix inversion.

5.3 Incremental RWR Computing

We now study the incremental RWR computation.

Given the old $\mathbf{P}$ for G, changes ΔG to G, query q, and $c\in(0,1)$, the goal is to compute $[\mathbf{\Delta P}]_{\star,q}$ for ΔG. The key idea of our approach is to maximally reuse the previous computation, by characterizing $[\mathbf{\Delta P}]_{\star,q}$ as a linear combination of the columns from the old $\mathbf{P}$.

The main result in this chapter is as follows.

Theorem 5.2. *Any node proximity of a given query can be incrementally computed in $O(1)$ time for each edge update.* □

To prove Theorem 5.2, we first consider unit edge update, and then devise an incremental algorithm for batch updates, with the desired complexity bound.

5.3.1 Unit Update

The update (insertion/deletion) of an edge from G may lead to the changes $[\mathbf{\Delta P}]_{\star,q}$ of the proximity. We incrementally compute $[\mathbf{\Delta P}]_{\star,q}$ based on the following.

Proposition 5.3. Given a query q, and the old proximity matrix $\mathbf{P}$ for G, if there is an edge insertion (i,j) into G, then the changes $[\mathbf{\Delta P}]_{\star,q}$ *w.r.t.* q can be computed as

$$[\mathbf{\Delta P}]_{\star,q} = \frac{(1-c)[\mathbf{P}]_{j,q}}{1-(1-c)[\mathbf{y}]_j}\cdot\mathbf{y} \text{ with} \tag{5.4}$$

$$\mathbf{y} = \begin{cases} \frac{1}{c}[\mathbf{P}]_{\star,i} & (d_j = 0) \\ \frac{1}{c(d_j+1)}([\mathbf{P}]_{\star,i} - \frac{1}{1-c}[\mathbf{P}]_{\star,j}) + \frac{1}{(1-c)(d_j+1)}\mathbf{e}_j & (d_j > 0) \end{cases}$$

where d_j is the in-degree of node j in the old G, and $[\mathbf{y}]_j$ is the j-th entry of vector $\mathbf{y}$.

If there is an edge deletion (i, j) from G, then $[\mathbf{\Delta P}]_{\star,q}$ can also be computed via Eq.(5.4) with $\mathbf{y}$ being replaced by

$$\mathbf{y} = \begin{cases} -\frac{1}{c}[\mathbf{P}]_{\star,i} & (d_j = 1) \\ \frac{1}{c(d_j-1)}(\frac{1}{1-c}[\mathbf{P}]_{\star,j} - [\mathbf{P}]_{\star,i}) - \frac{1}{(1-c)(d_j-1)}\mathbf{e}_j & (d_j > 1) \end{cases} \quad \square$$

As opposed to the traditional methods, *e.g.,* k-dash and B_LIN, that requires *matrix-vector multiplications* to compute new proximities via Eq.(5.2), Proposition 5.3 allows merely *vector scaling and additions* for efficiently computing $[\mathbf{\Delta P}]_{\star,q}$.

The proof of Proposition 5.3 is attained by combining the three following lemmas.

Lemma 5.4. Let $\mathbf{A}$ be the old transition matrix of G. If there is an edge insertion (i, j) into G, then the new transition matrix $\tilde{\mathbf{A}}$ is updated by

$$\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{a}\mathbf{e}_j^T \text{ with } \mathbf{a} = \begin{cases} \mathbf{e}_i & (d_j = 0) \\ \frac{1}{d_j+1}(\mathbf{e}_i - [\mathbf{A}]_{\star,j}) & (d_j > 0) \end{cases} \tag{5.5}$$

If there is an edge deletion (i, j) from G, then the new $\tilde{\mathbf{A}}$ is also updated as Eq.(5.5) with $\mathbf{a}$ being replaced by

$$\mathbf{a} = \begin{cases} \mathbf{e}_i & (d_j = 1) \\ \frac{1}{d_j-1}([\mathbf{A}]_{\star,j} - \mathbf{e}_i) & (d_j > 1) \end{cases} \quad \square \tag{5.6}$$

Proof. Due to space limits, we shall merely prove the insertion case. A similar proof holds for the deletion case.

(i) When $d_j = 0$, $[\mathbf{A}]_{\star,j} = \mathbf{0}$. Thus, for an inserted edge (i, j), $[\mathbf{A}]_{i,j}$ will be updated from 0 to 1, *i.e.,* $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{e}_i\mathbf{e}_j^T$.

(ii) When $d_j > 0$, all the nonzero entries of $[\mathbf{A}]_{\star,j}$ are $\frac{1}{d_j}$. Thus, for an inserted edge (i, j), we first update $[\mathbf{A}]_{i,j}$ from 0 to $\frac{1}{d_j}$, *i.e.,* $\mathbf{A} \Rightarrow \mathbf{A} + \frac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T$, and then change

all nonzero entries of $[\mathbf{A} + \frac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T]_{\star,j}$ from $\frac{1}{d_j}$ to $\frac{1}{d_j+1}$. Recall from the elementary matrix property that multiplying the j-th column of a matrix by $\alpha \neq 0$ can be accomplished by using $\mathbf{I} - (1-\alpha)\mathbf{e}_j\mathbf{e}_j^T$ as a right-hand multiplier on the matrix. Hence, scaling $[\mathbf{A} + \frac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T]_{\star,j}$ by $\alpha = \frac{d_j}{d_j+1}$ yields

$$\begin{aligned}\tilde{\mathbf{A}} &= \left(\mathbf{A} + \tfrac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T\right)\left(\mathbf{I} - (1 - \tfrac{d_j}{d_j+1})\mathbf{e}_j\mathbf{e}_j^T\right) \\ &= \mathbf{A} + \tfrac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T - \tfrac{1}{d_j+1}(\mathbf{A} + \tfrac{1}{d_j}\mathbf{e}_i\mathbf{e}_j^T)\mathbf{e}_j\mathbf{e}_j^T \\ &= \mathbf{A} + \tfrac{1}{d_j+1}(\mathbf{e}_i - [\mathbf{A}]_{\star,j})\mathbf{e}_j^T\end{aligned}$$

Combining (i) and (ii), Eq.(5.5) is derived. □

Lemma 5.4 suggests that each edge change will incur a rank-one update of $\mathbf{A}$. To see how the update of $\mathbf{A}$ affects the changes to $\mathbf{P}$, we have the following lemma.

Lemma 5.5. Let $\mathbf{P}$ be the old proximity matrix for G. If there is an edge update (i, j) to G, then the new proximity $\tilde{\mathbf{P}}$ *w.r.t.* a given query q is updated as

$$[\tilde{\mathbf{P}}]_{\star,q} = [\mathbf{P}]_{\star,q} + (1-c)\gamma \cdot \mathbf{z} \tag{5.7}$$

$$\text{with } \gamma = \tfrac{[\mathbf{P}]_{j,q}}{1-(1-c)\cdot[\mathbf{z}]_j} \text{ and } \mathbf{z} = (\mathbf{I} - (1-c)\mathbf{A})^{-1}\mathbf{a},$$

where the vector $\mathbf{a}$ is defined by Lemma 5.4. □

Proof. By RWR definition in Eq.(5.3), $[\tilde{\mathbf{P}}]_{\star,q}$ satisfies

$$(\mathbf{I} - (1-c)\tilde{\mathbf{A}}) \cdot [\tilde{\mathbf{P}}]_{\star,q} = c\mathbf{e}_q \tag{5.8}$$

where $\tilde{\mathbf{A}}$ is the new transition matrix that is expressed as $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{a}\mathbf{e}_j^T$ by Lemma 5.4. Thus, Eq.(5.8) is rewritten as

$$\begin{bmatrix} \mathbf{I} - (1-c)\mathbf{A} & -(1-c)\mathbf{a} \\ \mathbf{e}_j^T & -1 \end{bmatrix} \begin{bmatrix} [\tilde{\mathbf{P}}]_{\star,q} \\ \gamma \end{bmatrix} = \begin{bmatrix} c\mathbf{e}_q \\ 0 \end{bmatrix} \tag{5.9}$$

To solve $[\tilde{\mathbf{P}}]_{\star,q}$ and γ in Eq.(5.9), we apply block elimination, by using block elementary row operations, and starting with the associated augmented matrix:

$$\left[\begin{array}{cc|c} \mathbf{I} - (1-c)\mathbf{A} & -(1-c)\mathbf{a} & c\mathbf{e}_q \\ \mathbf{e}_j^T & -1 & 0 \end{array}\right] \xrightarrow{\text{Row2}-\mathbf{e}_j^T(\mathbf{I}-(1-c)\mathbf{A})^{-1}\cdot\text{Row1}}$$

$$\rightarrow \left[\begin{array}{cc|c} \mathbf{I}-(1-c)\mathbf{A} & -(1-c)\mathbf{a} & c\mathbf{e}_q \\ 0 & (1-c)\mathbf{e}_j^T(\mathbf{I}-(1-c)\mathbf{A})^{-1}\mathbf{a}-1 & -c\mathbf{e}_j^T[\mathbf{P}]_{\star,q} \end{array}\right]$$

The final array represents the following equations:

$$\begin{cases} (\mathbf{I}-(1-c)\mathbf{A})\,[\tilde{\mathbf{P}}]_{\star,q} - \gamma(1-c)\mathbf{a} = c\mathbf{e}_q \\ \left((1-c)\mathbf{e}_j^T(\mathbf{I}-(1-c)\mathbf{A})^{-1}\mathbf{a}-1\right)\gamma = -c[\mathbf{P}]_{j,q} \end{cases}$$

Back substitution, along with Eq.(5.3), yields Eq.(5.7). □

Lemma 5.5 tells that for each edge update, the changes to $\mathbf{P}$ are just associated with the scaling operation of vector $\mathbf{z}$. However, it is costly to compute $\mathbf{z}$ via Eq.(5.7) as it involves the inversion of a matrix. Lemma 5.6 provides an efficient way of computing $(\mathbf{I}-(1-c)\mathbf{A})^{-1}\mathbf{a}$ from a few columns of $\mathbf{P}$.

Lemma 5.6. Suppose there is an edge update (i,j) to G, and $\mathbf{a}$ is defined by Lemma 5.4. Then, $(\mathbf{I}-(1-c)\mathbf{A})^{-1}\mathbf{a} = \mathbf{y}$ with $\mathbf{y}$ being defined in Proposition 5.3. □

Proof. Due to space limits, we shall only prove the edge insertion case. A similar proof holds for the deletion case.

(i) When $d_j = 0$, $\mathbf{a} = \mathbf{e}_i$. Then, Eq.(5.3) implies that

$$(\mathbf{I}-(1-c)\mathbf{A})^{-1}\mathbf{e}_i = \tfrac{1}{c}[\mathbf{P}]_{\star,i}$$

(ii) When $d_j > 0$, $\mathbf{a} = \frac{1}{d_j+1}(\mathbf{e}_i - [\mathbf{A}]_{*,j})$. Then,

$$\begin{aligned}(\mathbf{I}-(1-c)\mathbf{A})^{-1}\mathbf{a} &= \tfrac{1}{d_j+1}(\mathbf{I}-(1-c)\mathbf{A})^{-1}(\mathbf{e}_i - [\mathbf{A}]_{\star,j}) \\ &= \tfrac{1}{d_j+1}(\tfrac{1}{c}[\mathbf{P}]_{\star,i} - (\mathbf{I}-(1-c)\mathbf{A})^{-1}[\mathbf{A}]_{\star,j})\end{aligned}$$

To solve $(\mathbf{I}-(1-c)\mathbf{A})^{-1}[\mathbf{A}]_{\star,j}$, we apply the property that $(\mathbf{I}-\mathbf{X})^{-1} = \sum_{k=0}^{\infty}\mathbf{X}^k$ (for $\|\mathbf{X}\|_1 < 1$) and obtain

$$\begin{aligned}(\mathbf{I}-(1-c)\mathbf{A})^{-1}\mathbf{A} &= \sum_{k=0}^{\infty}(1-c)^k\mathbf{A}^{k+1} \\ &= \tfrac{1}{1-c}\sum_{k=1}^{\infty}(1-c)^k\mathbf{A}^k \\ &= \tfrac{1}{1-c}((\mathbf{I}-(1-c)\mathbf{A})^{-1}-\mathbf{I})\end{aligned}$$

$$= \frac{1}{1-c}(\frac{1}{c}\mathbf{P} - \mathbf{I})$$

Thus, we have $(\mathbf{I} - (1-c)\mathbf{A})^{-1}[\mathbf{A}]_{\star,j} = \frac{1}{1-c}(\frac{1}{c}[\mathbf{P}]_{\star,j} - \mathbf{e}_j)$. Substituting this back produces the final results. □

Combining Lemmas 5.4–5.6 together proves Proposition 5.3.

5.3.2 Batch Update

Based on Proposition 5.3, we devise IRWR, an incremental RWR algorithm to handle a set ΔG of edge insertions and deletions (batch update).

IRWR is shown in Algorithm 5.1. Given the old $\mathbf{P}$ for G *w.r.t.* query q, and the batch edge updates ΔG, it computes new proximities *w.r.t.* q in $G \cup \Delta G$ without loss of exactness. It works as follows. For each edge (i, j) to be updated, it first computes the auxiliary vector $\mathbf{y}$ from a linear combination of only a few columns in $\mathbf{P}$ (lines 2–10). Using $\mathbf{y}$, it then (i) removes (i, j) from ΔG (line 11) and (ii) updates the proximities *w.r.t.* each remaining node in ΔG (lines 12–14). After all the edges are eliminated from ΔG, IRWR finally calculates the new proximities $[\tilde{\mathbf{P}}]_{\star,q}$ from $\mathbf{y}$ (line 15).

Example 5.7. Recall $\mathbf{P}$ and G of Figure 5.1. Consider batch updates ΔG, which insert edge (u_1, u_5) and delete (u_4, u_6), where (u_1, u_5) is given in Example 5.1. IRWR computes the new proximities $[\mathbf{P}]_{\star,u_2}$ *w.r.t.* query u_2 in $G + \Delta G$ as follows:

For the edge insertion (u_1, u_5), since $d_{u_5} = 3$ and $c = 0.2$, $\mathbf{y} = 1.25 \times [\mathbf{P}]_{\star,u_1} - 1.56 \times [\mathbf{P}]_{\star,u_5} + 0.31 \times \mathbf{e}_5$ (via line 5).

Before proceeding with the edge deletion, let us look at the changes $[\mathbf{\Delta P}]_{\star,u_2}$ (via line 14) for the inserted (u_1, u_5):

$$\begin{aligned}[\mathbf{\Delta P}]_{\star,u_2} &= (1-c)\gamma \cdot \mathbf{y} \quad \text{with } \gamma = \frac{[\mathbf{P}]_{u_5,u_2}}{1-(1-c)[\mathbf{y}]_5} = 0.254 \\ &= 0.25 \times [\mathbf{P}]_{\star,u_1} - 0.32 \times [\mathbf{P}]_{\star,u_5} + 0.06 \times \mathbf{e}_5,\end{aligned}$$

which explains why the values of α, β, λ are chosen for Eq.(5.1).

IRWR then removes (u_1, u_5) from ΔG (line 11). Using $\mathbf{y}$, it updates proximities *w.r.t.* $u_4, u_6 \in \Delta G$ (lines 12–14). Thus, $[\mathbf{P}]_{\star,u_4} = (.17, .05, .23, .28, .23, .05, 0)^T$, and

Algorithm 5.1: IRWR $(G, \mathbf{P}, q, \Delta G, c)$

Input : graph G, old proximities $\mathbf{P}$ for G, query node q, updates ΔG to G, and restarting probability c.

Output: new proximities $[\tilde{\mathbf{P}}]_{\star,q}$ *w.r.t.* q.

1 **foreach** *edge* $(i,j) \in \Delta G$ *to be updated* **do**
2 $\quad d_j :=$ in-degree of node j in G ;
3 $\quad$ **if** *edge* (i,j) *is to be inserted* **then**
4 $\quad\quad$ **if** $d_j = 0$ **then** $\mathbf{y} := \frac{1}{c}[\mathbf{P}]_{\star,i}$;
5 $\quad\quad$ **else** $\mathbf{y} := \frac{1}{d_j+1}\big(\frac{1}{c}([\mathbf{P}]_{\star,i} - \frac{1}{1-c}[\mathbf{P}]_{\star,j}) + \frac{1}{(1-c)}\mathbf{e}_j\big)$;
6 $\quad\quad G := G \cup \{(i,j)\}$;
7 $\quad$ **else if** *edge* (i,j) *is to be deleted* **then**
8 $\quad\quad$ **if** $d_j = 1$ **then** $\mathbf{y} := -\frac{1}{c}[\mathbf{P}]_{\star,i}$;
9 $\quad\quad$ **else** $\mathbf{y} := \frac{1}{d_j-1}\big(\frac{1}{c}(\frac{1}{1-c}[\mathbf{P}]_{\star,j} - [\mathbf{P}]_{\star,i}) - \frac{1}{(1-c)}\mathbf{e}_j\big)$;
10 $\quad\quad G := G \backslash \{(i,j)\}$;
11 $\quad \Delta G := \Delta G \backslash \{(i,j)\}$;
12 $\quad$ **if** $\Delta G \neq \varnothing$ **then**
13 $\quad\quad$ **foreach** $v \in \{$*vertices in* $\Delta G\} \cup \{q\}$ **do**
14 $\quad\quad\quad \gamma := \frac{[\mathbf{P}]_{j,v}}{1-(1-c)[\mathbf{y}]_j}$, $[\mathbf{P}]_{\star,v} := [\mathbf{P}]_{\star,v} + (1-c)\gamma\mathbf{y}$;
15 $\quad$ **else** $\gamma := \frac{[\mathbf{P}]_{j,q}}{1-(1-c)[\mathbf{y}]_j}$, $[\tilde{\mathbf{P}}]_{\star,q} := [\mathbf{P}]_{\star,q} + (1-c)\gamma\mathbf{y}$;
16 **return** $[\tilde{\mathbf{P}}]_{\star,q}$;

$[\mathbf{P}]_{\star,u_6} = (.14, .04, .18, .23, .18, .24, 0)^T$ after $(u1, u5)$ is added to G.

Likewise, for the edge deletion (u_4, u_6), $d_{u_6} = 1$ implies $\mathbf{y} = -\frac{1}{0.2} \times [\mathbf{P}]_{\star,u_4}$ (line 8). Then, (u_4, u_6) is removed from ΔG (line 11). Since $\Delta G = \varnothing$, the changes $[\mathbf{\Delta P}]_{\star,u_2}$ for the deleted (u_4, u_6) is obtained (via line 15):

$$[\mathbf{\Delta P}]_{\star,u_2} = 0.8\gamma \cdot \mathbf{y} = -0.17 \times [\mathbf{P}]_{\star,u_4} \text{ with } \gamma = \frac{[\mathbf{P}]_{u_6,u_2}}{1-(1-c)[\mathbf{y}]_6} = 0.04$$

$$\Rightarrow [\tilde{\mathbf{P}}]_{\star,u_2} = [\mathbf{P}]_{\star,u_2} + [\mathbf{\Delta P}]_{\star,u_2} = (.25, .24, .04, .04, .22, .04, 0)^T. \quad \square$$

Correctness & Complexity. To complete the proof of Theorem 5.2, we notice that (i) IRWR can correctly compute RWR proximities, which is verified by Proposition 5.3. Moreover, IRWR always terminates, since the size of ΔG is monotonically decreasing. (ii) One can readily verify that for each edge update, IRWR involves only vector scaling and additions, which is in $O(1)$ time for each node proximity.

5.4 Experimental Evaluation

We present an empirical study on real and synthetic data to evaluate the efficiency of IRWR for incremental computation, as compared with (a) its batch counterpart B_LIN [TFP06], (b) k-dash [FNOK12], the best known algorithm for top-k search, and (c) IncPPR [BCG10], the incremental personalized PageRank.

5.4.1 Experimental Setting

We use both real and synthetic datasets.

Two real datasets are adopted: (a) p2p-Gnutella, a Gnutella P2P digraph, in which nodes represent hosts, and edges host connections. The dataset has 62.5K nodes and 147.9K edges. (b) cit-HepPh, a citation network from Arxiv, where nodes denote papers, and edges paper citations. We extracted a snapshot with 27.7K nodes and 352.8K edges.

GraphGen[3] is used to build synthetic graphs and updates. Graphs are controlled by (a) the number of nodes $|V|$, and (b) the number of edges $|E|$; updates by (a) update type (edge insertion or deletion), and (b) the size of updates $|\Delta G|$.

All the algorithms are implemented in Visual C++ 10.0. We used a machine with an Intel Core(TM) 3.10GHz CPU and 8GB RAM, running Windows 7. Each experiment is run 5 times. We report the average here.

We set the restarting probability $c = 0.2$ in our experiments.

[3] http://www.cse.ust.hk/graphgen/

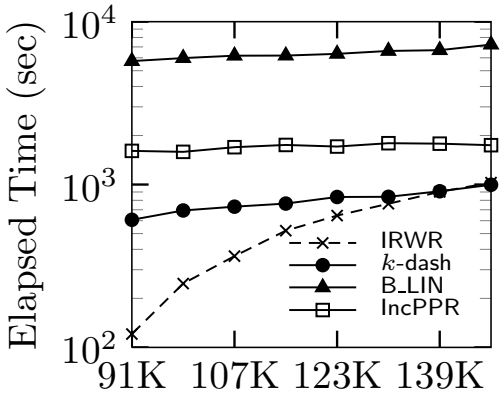


Figure 5.2: IRWR on p2p-Gnutella

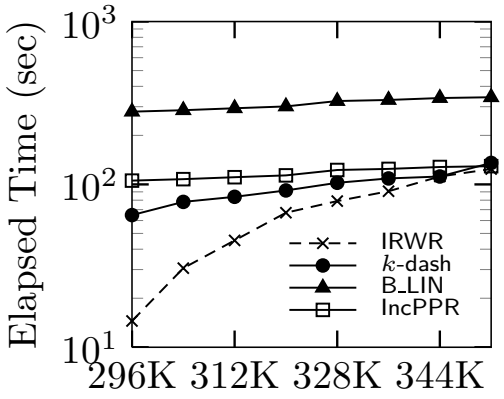


Figure 5.3: IRWR on cit-HepPh

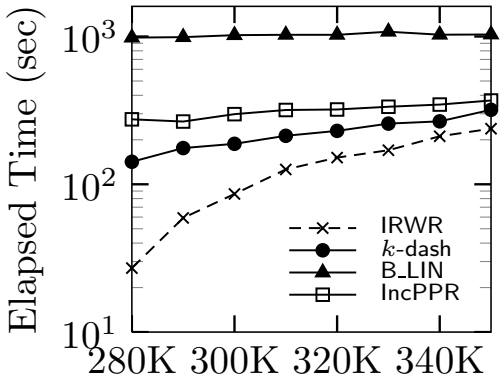


Figure 5.4: Edge insertions

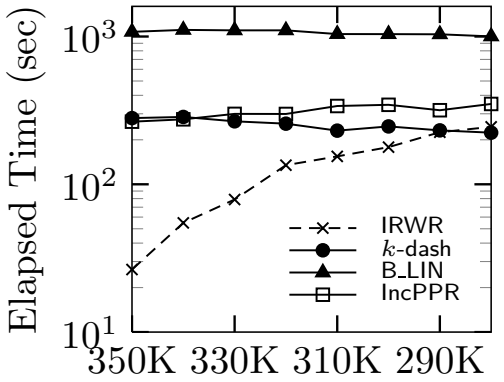


Figure 5.5: Edge deletions

5.4.2 Experimental Results

We next present our evaluation results.

Exp-1: Incremental Efficiency

We first evaluate the computational time of IRWR on both real and synthetic data.

Figures 5.2 and 5.3 depict the results for edges inserted to p2p-Gnutella ($|V|$=62.5K) and cit-HepPh ($|V|$=27.7K), respectively. Fixing $|V|$, we vary $|E|$ as shown in the x-axis. Here, the updates are the difference of snapshots *w.r.t.* the collection time of datasets, reflecting their real-life evolution. We find that (a) IRWR outperforms k-dash on p2p-Gnutella for 92.7% (*resp.*cit-HepPh for 97.5%) of edge updates. When the changes are 61.9% on p2p-Gnutella (83.8% on cit-HepPh), IRWR improves k-dash by over 5.1x (*resp.*4.4x). This is because IRWR reuses the old information in G for incrementally

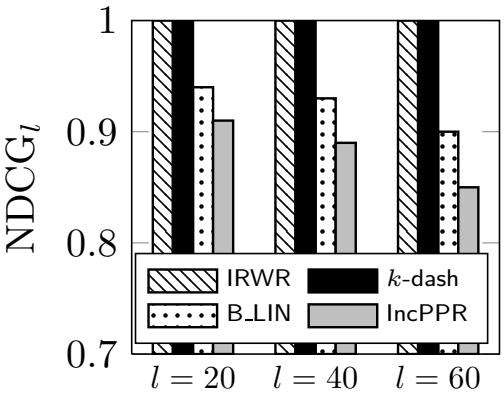


Figure 5.6: Exactness on p2p-Gnutella

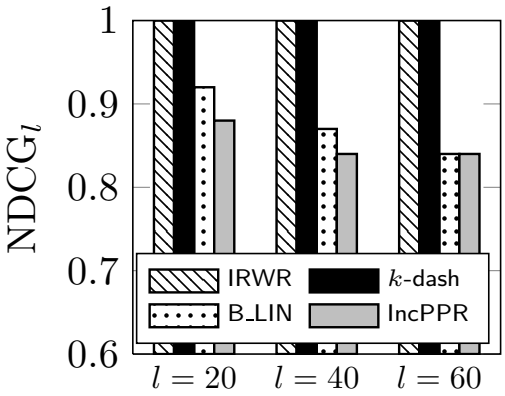


Figure 5.7: Exactness on cit-HepPh

updating proximities via vector scaling and additions, without the need for expensive LU decomposition of k-dash. (b) IRWR always is better than B_LIN by nearly one order of magnitude as B_LIN requires costly block matrix inversions. (c) IRWR outperforms IncPPR for over 95% of insertions, due to the extra cost of IncPPR for doing short random walks. (d) IRWR is sensitive to $|\Delta G|$ as the larger $|\Delta G|$ is, the larger the affected area is, so is the computation cost, as expected.

Fixing $|V|$=50K on synthetic data, we varied $|E|$ from 280K to 350K (*resp.*from 350K to 280K) in 10K increments (*resp.*decrements). The results are shown in Figures 5.4 and 5.5, respectively, analogous to those on real datasets.

Exp-2: Exactness

To measure IRWR accuracy, we adopted NDCG$_l$ (Normalized Discounted Cumulative Gain) for ranking top-l node proximities with $l = 20, 40, 60$, and chose the ranking results of k-dash as the benchmark, due to its exactness. The results on p2p-Gnutella and cit-HepPh are reported in Figures 5.6 and 5.7, indicating that IRWR never scarifies accuracy for achieving high efficiency, superior to other approaches.

5.5 Related Work

Incremental algorithms have proved useful in various node proximity computations on evolving graphs, such as the personalized PageRank [BCG10] and SimRank [LHH+10].

However, very few results are known on incremental RWR computing, far less than their batch counterparts [TFP06, ZCY09, FNOK12]. k-dash [FNOK12] is the best known approach to finding top-k highest RWR proximity nodes for a given query, which involves a strategy to incrementally *estimate* upper proximity bounds. Nevertheless, such an incremental strategy is *approximate*: in $O(1)$ time for each node, which is mainly developed for pruning unnecessary computation. In contrast, our incremental algorithm can, without loss of exactness, compute any node proximity in $O(1)$ time for every edge update. Moore *et al.* [SMP08] leveraged a sampling approach with branch and bound pruning to find near neighbors of a query *w.h.p.*. However, their incremental algorithm is *probabilistic*. Later, Zhou *et al.* [ZCY09] generalized the original RWR by incorporating node attributes into link structure for graph clustering. Based on this, an incremental version of [ZCY09] was proposed by Cheng *et al.* [CZHY12], with the focus to support *attribute update*. It differs from this work in that our incremental algorithm is designed for *structure update*. Thus, [CZHY12] cannot cope with *hyperlink changes* incrementally in dynamic graphs.

5.6 Conclusions

In the chapter, we showed how RWR proximities can be computed very efficiently in an incremental update model, where the edges of a graph are constantly changed. First, we focused on unit update by obtaining an elegant formula that characterizes the RWR proximity changes as a linear combination of the columns from the old proximity matrix. Then, we proposed an incremental algorithm for batch update, by showing that any node proximity can be computed in $O(1)$ time for every edge update, with no sacrifice in accuracy. Finally, we also empirically verified that IRWR greatly outperforms the other approaches on both real and synthetic graphs without loss of exactness. As a future avenue, we will further predict up to what fraction of updated edges IRWR is faster than its batch counterparts.

Chapter 6

Fast SimFusion+ on Large and Dynamic Networks

6.1 Introduction

The conundrum of measuring similarity between objects based on hyperlinks in a graph has fueled a growing interest in the fields of information retrieval. Recently, while the scale of the Web has dramatically increased our need to produce large graphs, the study of efficiently computing object similarity on such large graphs becomes a desideratum.

Among the existing metrics, SimFusion [XFF+05] can be regarded as one of the attractive ones on account of the following reasons.

(1) Similar to PageRank [PBMW99] and SimRank [JW02], SimFusion is based on hyperlinks and follows the reinforcement assumption that "the similarity between objects is reinforced by the similarity of their related objects", which is fairly intuitive and conforms to our basic understandings.

(2) Unlike other measures (*e.g.,* PageRank and SimRank) that explore the linkage patterns merely from a single data space [PBMW99, ZHS09, JW02], SimFusion has the extra benefits of incorporating both inter- and intra-relationships from multiple data spaces in a unified manner to measure the similarity of heterogeneous data objects.

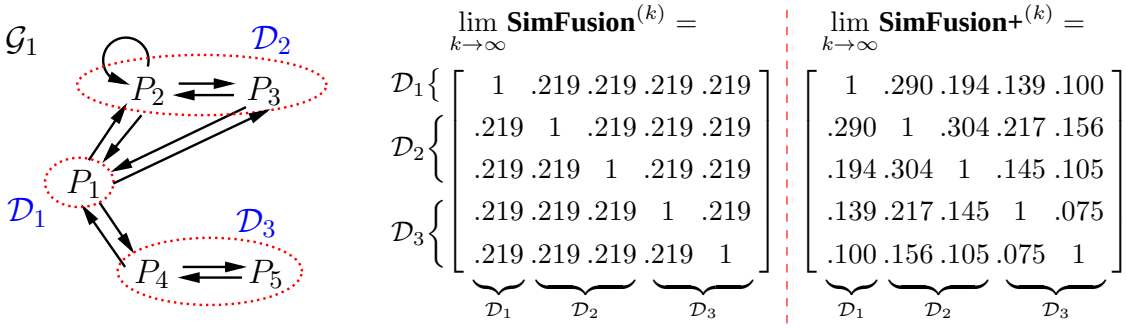


Figure 6.1: Trivial SimFusion on Heterogeneous Domain

(3) SimFusion offers more intuitive and flexible ways of assigning weighting factors to each data space that reflects their relative importance, as opposed to the PageRank and SimRank measures that need to determine a damping factor.

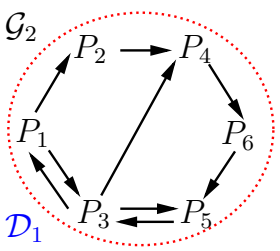
(4) SimFusion provides a general-purpose framework for measuring structural similarity in a recursive fashion; other well-known measures, such as Co-Citation [Sma73] and Coupling [Jar07] are just special cases of SimFusion.

6.1.1 Motivation

Despite the aforementioned merits, existing work on SimFusion has the following problems, motivating us for an in-depth investigation.

Firstly, although the basic intuition behind the SimFusion model is appealing, it seems inappropriate to use the *Unified Relationship Matrix* (URM) to represent the relationships of heterogeneous objects. The main problem is that, according to the definition of URM $\mathbf{L}$ in [XFF+05], the sum of each row of $\mathbf{L}$ is always equal to 1. Since the product of $\mathbf{L}$ and the matrix $\mathbf{1}$ whose entries are all ones is equal to the matrix $\mathbf{1}$ of all ones, there always exists a trivial solution $\mathbf{S} = \mathbf{1}$ to the original SimFusion formula $\mathbf{S} = \mathbf{L} \cdot \mathbf{S} \cdot \mathbf{L}^T$ [XFF+05], as illustrated in Example 6.1. The same phenomena of yielding such a trivial solution may occur in our experimental results in Section 6.6. To address this issue, we shall revise the original SimFusion model.

Example 6.1 (Trivial Solution)**.** Figure 6.1 depicts a graph $\mathcal{G}_1$ partly extracted from Cornell CS Department. Each vertex P_i denotes a web page, and each edge a hyperlink.



$$\lim_{k\to\infty} \textbf{SimFusion}^{(2k)} = \begin{bmatrix} .46 & 0 & .46 & 0 & 0 & .46 \\ 0 & .38 & 0 & .38 & .38 & 0 \\ .46 & 0 & .46 & 0 & 0 & .46 \\ 0 & .38 & 0 & .38 & .38 & 0 \\ 0 & .38 & 0 & .38 & .38 & 0 \\ .46 & 0 & .46 & 0 & 0 & .46 \end{bmatrix} \neq \lim_{k\to\infty} \textbf{SimFusion}^{(2k+1)} = \begin{bmatrix} .38 & 0 & .38 & 0 & 0 & .38 \\ 0 & .46 & 0 & .46 & .46 & 0 \\ .38 & 0 & .38 & 0 & 0 & .38 \\ 0 & .46 & 0 & .46 & .46 & 0 \\ 0 & .46 & 0 & .46 & .46 & 0 \\ .38 & 0 & .38 & 0 & 0 & .38 \end{bmatrix}$$

$$\lim_{k\to\infty} \textbf{SimFusion+}^{(k)} = \begin{bmatrix} 1 & 0 & .25 & 0 & 0 & .25 \\ 0 & 1 & 0 & .25 & .18 & 0 \\ .25 & 0 & 1 & 0 & 0 & .18 \\ 0 & .25 & 0 & 1 & .25 & 0 \\ 0 & .18 & 0 & .25 & 1 & 0 \\ .25 & 0 & .18 & 0 & 0 & 1 \end{bmatrix}$$

Figure 6.2: Divergent SimFusion on Homogeneous Domain

There are three categories: $\mathcal{D}_1 = \{P_1\}$ (student), $\mathcal{D}_2 = \{P_2, P_3\}$ (staff), and $\mathcal{D}_3 = \{P_4, P_5\}$ (faculty). We want to retrieve the top-3 similar pairs of web pages in $\mathcal{G}_1$. However, the naive SimFusion fails to correctly find them. We observe that the SimFusion solution is a (trivial) matrix whose entries are all the same. In fact, vertices in $\mathcal{G}_1$ do not have the identical neighbor structures. Hence, the trivial solution is non-semantic in real communities. □

Secondly, it is rather expensive to compute SimFusion similarities. The existing approach for SimFusion computation deploys a fixed-point iteration: $\mathbf{S}^{(k+1)} = \mathbf{L} \cdot \mathbf{S}^{(k)} \cdot \mathbf{L}^T$, which requires $O(kn^3)$ time and $O(n^2)$ space [XFF+05]. This impedes the scalability of SimFusion on large graphs. Worse still, the iterative computation of SimFusion do not always converge. The convergence of the SimFusion iterations heavily depends on the choice of the initial guess $\mathbf{S}^{(0)}$, as shown in Example 6.2.

Example 6.2 (Divergence SimFusion)**.** Consider the disease transmission graph $\mathcal{G}_2$, where each vertex is an organism P_i which can carry the disease, and an edge represents one organism spreading it to another. One wants to find the three most similar organisms to P_2 in $\mathcal{G}_2$. However, the iterative computation of SimFusion does not work properly. We observe the following:

(i) When $\mathbf{S}^{(0)}$ is set to an $n \times n$ identity matrix (according to [XFF+05]) the "even" and "odd" subsequences of $\{\mathbf{S}^{(k)}\}$ are convergent respectively, but they do not converge to the same limit, which makes the full sequence $\{\mathbf{S}^{(k)}\}$ divergent.

(ii) Choosing $\mathbf{S}^{(0)} = \mathbf{1}_n$ (*i.e.,* an $n \times n$ matrix of all 1s) instead, we observe that the full SimFusion sequence $\{\mathbf{S}^{(k)}\}$ is always convergent to $\mathbf{1}_n$ regardless of the graph

structure. □

This suggests that the original SimFusion iterations may be divergent or converge to a trivial solution, not to mention its scalability. This highlights the need to find a feasible way to guarantee the convergence of the SimFusion iterations, but it is hard to devise an efficient algorithm for the revised SimFusion computation.

Thirdly, it is a big challenge to incrementally compute SimFusion on dynamic graphs. The traditional method [XFF+05] has to recompute the similarity from scratch when edges in a graph change over time, which is not adaptive to many evolving networks. Fortunately, we have an observation that the size of the areas affected by the updates is typically small in practice. To this end, we propose an incremental algorithm that fully utilizes these affected areas to compute SimFusion on dynamic graphs.

6.1.2 Chapter Outlines

In this chapter, we propose SimFusion+, a revised notion of SimFusion, to provide a full treatment of SimFusion for the convergence issues and to improve its computational efficiency. In summary, we make the following contributions.

- We formalize the problem of SimFusion+ computation (Section 6.2). The notion of SimFusion+ revises the divergence and non-semantic convergence worries of the traditional model [XFF+05].

- We present optimization techniques for improving the computation of SimFusion+ to $O(1)$ time and $O(n)$ space for every pair of vertices, plus an $O(km)$-time pre-computation run only once (Section 6.3).

- We devise an efficient algorithm to compute the SimFusion+ similarity with better accuracy guarantees (Section 6.4). An error estimate is also given for the SimFusion+ approximation.

- We devise an incremental algorithm for further optimizing the SimFusion+ computation when edges in networks are dynamically updated (Section 6.5). We show

that the update cost of the incremental algorithm retains $O(\delta n)$ time and $O(n)$ space for handling a sequence of δ edge insertions or deletions.

- We experimentally verify the effectiveness and scalability of the algorithms, using real and synthetic data (Section 6.6). The results show that SimFusion+ can govern the convergence towards a meaningful solution, and our algorithms achieve high accuracy and significantly outperform the baseline algorithms.

6.2 SimFusion Estimation Revised

In this section, we first revisit the definition of data space and data relation. We then introduce the notion of the *Unified Adjacency Matrix* (UAM) to revise the SimFusion model.

6.2.1 Data Space and Data Relation

Graphs studies here are directed graphs having no multiple edges.

Data Space. A *data space* is the finite set of all data objects (vertices) with the same data type, denoted by $\mathcal{D} = \{o_1, o_2, \cdots\}$. $|\mathcal{D}|$ denotes the number of data objects in $\mathcal{D}$. Two nonempty data spaces $\mathcal{D}$ and $\mathcal{D}'$ are said to be *disjoint* if $\mathcal{D} \cap \mathcal{D}' = \varnothing$.

Throughout the paper, we shall use the following notations. (i) The *entire space* $\mathcal{D}$ in a network is the union of N disjoint data spaces $\mathcal{D}_1, \cdots, \mathcal{D}_N$ such that $\mathcal{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$ and $\mathcal{D}_i \cap \mathcal{D}_j = \varnothing \quad (i \neq j)$. (ii) The total size $|\mathcal{D}|$ of the entire space, denoted by n, is the sum of the number n_i of the data objects in each data space $\mathcal{D}_i$, *i.e.*, $n = \sum_{i=1}^{N} n_i$ with $n_i = |\mathcal{D}_i| \quad (\forall i = 1, \cdots, N)$.

Intuitively, for heterogeneous networks, the distinct spaces $\mathcal{D}_i$ form a partition of $\mathcal{D}$ into classes. For homogenous networks, the partition of $\mathcal{D}$ is itself.

Data Relation. A *data relation* on $\mathcal{D}$ is defined as $\mathcal{R} \subseteq \mathcal{D} \times \mathcal{D}$, where $(o, o') \in \mathcal{R}$ is a connection (a directed edge) from object o to o'. Data objects in the same data space are related via *intra-type relations* $\mathcal{R}_{i,i} \subseteq \mathcal{D}_i \times \mathcal{D}_i$. Data objects between distinct data

spaces are related via *inter-type relations* $\mathcal{R}_{i,j} \subseteq \mathcal{D}_i \times \mathcal{D}_j \quad (i \neq j)$.

Intuitively, the intra-type relation carries connected information in each data space (*e.g.,* co-citation between web pages); the inter-type relation represents interlinked information between different data spaces (*e.g.,* making user requests). As an example, in Figure 6.1 there are three data spaces : $\mathcal{D} = \{P_1\} \cup \{P_2, P_3\} \cup \{P_4, P_5\}$, where

(a) $(P_2, P_2), (P_2, P_3), (P_3, P_2), (P_4, P_5), (P_5, P_4)$ are intra-relations;

(b) $(P_1, P_2), (P_2, P_1), (P_3, P_1), (P_1, P_3), (P_1, P_4), (P_4, P_1)$ are inter-relations.

6.2.2 Unified Adjacency Matrix

Let us now introduce the *unified adjacency matrix* (UAM). Consider a graph $\mathcal{G} = (\mathcal{D}, \mathcal{R})$ with data space $\mathcal{D}$ and data relation $\mathcal{R}$.

Unified Adjacency Matrix (UAM). The matrix $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{1}/n^2$ of size $n \times n$ is said to be a *unified adjacency matrix* of the relation $\mathcal{R}$ whenever

$$\tilde{\mathbf{A}} = \begin{pmatrix} \lambda_{1,1}\mathbf{A}_{1,1} & \lambda_{1,2}\mathbf{A}_{1,2} & \cdots & \lambda_{1,N}\mathbf{A}_{1,N} \\ \lambda_{2,1}\mathbf{A}_{2,1} & \lambda_{2,2}\mathbf{A}_{2,2} & \cdots & \lambda_{2,N}\mathbf{A}_{2,N} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{N,1}\mathbf{A}_{N,1} & \lambda_{N,2}\mathbf{A}_{N,2} & \cdots & \lambda_{N,N}\mathbf{A}_{N,N} \end{pmatrix},$$

where (i) $\mathbf{1}$ is the $n \times n$ matrix of all ones; (ii) $\mathbf{A}_{i,j}$ is the submatrix of $\mathbf{A}$ whose (o, o')-entry equals 1 if there is an edge from data object o to o', *i.e.,* $\exists\,(o, o') \in \mathcal{R}$, $\frac{1}{n_j}$ if data object o has no neighbors in $\mathcal{D}_j$, or 0 otherwise; and (iii) $\lambda_{i,j}$ is called the *weighting factor* between data space $\mathcal{D}_i$ and $\mathcal{D}_j$ with $0 \leq \lambda_{i,j} \leq 1$ and $\sum_{j=1}^{N} \lambda_{i,j} = 1$ $(\forall i = 1, \cdots, N)$.

Intuitively, $\mathbf{A}_{i,j}$ represents the intra- $(i = j)$ or inter- $(i \neq j)$ relation from data space $\mathcal{D}_i$ to $\mathcal{D}_j$. $\lambda_{i,j}$ reflects the relative importance between data spaces $\mathcal{D}_i$ and $\mathcal{D}_j$.

Example 6.3. In Figure 6.1, the relative importance between data space $\mathcal{D}_i$ and $\mathcal{D}_j$ is denoted by a weighting matrix $\mathbf{\Lambda} = (\lambda_{i,j})_{3\times 3}$. Then, the UAM $\mathbf{A}$ of $\mathcal{G}_1$ can be derived

from $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{1}/n^2$, where $\tilde{\mathbf{A}}$ is computed from $\mathbf{\Lambda}$ as follows.

$$\mathbf{\Lambda} = \begin{array}{c} \\ \mathcal{D}_1 \\ \mathcal{D}_2 \\ \mathcal{D}_3 \end{array} \begin{array}{c} \begin{array}{ccc} \mathcal{D}_1 & \mathcal{D}_2 & \mathcal{D}_3 \end{array} \\ \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{3} \\ \frac{1}{6} & \frac{7}{12} & \frac{1}{4} \\ \frac{1}{3} & \frac{1}{4} & \frac{5}{12} \end{bmatrix} \end{array} \Rightarrow \tilde{\mathbf{A}} = \begin{bmatrix} \frac{1}{2}[1] & \frac{1}{6}[1\ 1] & \frac{1}{3}[1\ 0] \\ \frac{1}{6}\begin{bmatrix} 1 \\ 1 \end{bmatrix} & \frac{7}{12}\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} & \frac{1}{4}\begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} \\ \frac{1}{3}\begin{bmatrix} 1 \\ 0 \end{bmatrix} & \frac{1}{4}\begin{bmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix} & \frac{5}{12}\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \end{bmatrix} = \left[\begin{array}{c|cc|cc} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & 0 \\ \hline \frac{1}{6} & \frac{7}{12} & \frac{7}{12} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{6} & \frac{7}{12} & 0 & \frac{1}{8} & \frac{1}{8} \\ \hline \frac{1}{3} & \frac{1}{8} & \frac{1}{8} & 0 & \frac{5}{12} \\ 0 & \frac{1}{8} & \frac{1}{8} & \frac{5}{12} & 0 \end{array}\right] \quad \square$$

SimFusion+ Model. In light of the UAM $\mathbf{A}$, we next propose the revised model of SimFusion, termed SimFusion+, as follows:

$$\mathbf{S} = \frac{\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T}{\|\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T\|_2}, \tag{6.1}$$

where $\mathbf{S}$ is called the *Unified Similarity Matrix* (USM) whose (i, j)-entry represents the similarity score between object i and j.

The uniqueness and existence of the SimFusion+ solution $\mathbf{S}$ to Eq.(6.1) can be established by *the power iteration* [Wil07, pp.381]. A detailed proof will be shown in Proposition 6.5 (Section 6.3).

The revised notion of SimFusion utilizes UAM (rather than URM) to represent data relations because UAM can effectively avoid divergent and trivial similarity solutions while well preserving the intuitive reinforcement assumption of the original model [XFF+05]. We observe that the root cause of the flawed solution to the original SimFusion is the "row normalization" of URM. Thus, by using UAM, we have an opportunity to postpone the operation of "row normalization" in a delayed fashion. To this end, we utilize the matrix 2-norm $\|\mathbf{A} \cdot \mathbf{S} \cdot \mathbf{A}^T\|_2$ to squeeze similarity scores in $\mathbf{S}$ into $[0, 1]$. The obtained similarity results in USM can not only prevent the divergence issue and the trivial solution but effectively capture the reliability of the similarity evidence between data objects. For instance, the SimFusion+ USMs in Examples 6.1 and 6.2 are nontrivial and intuitively explainable.

6.3 Computing Similarity Via Dominant Eigenvector

A conventional approach for finding the SimFusion+ solution $\mathbf{S}$ to Eq.(6.1) is to employ the following fixed-point iteration: [1]

$$\mathbf{S}^{(k+1)} = \frac{\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T}{\left\|\mathbf{A} \cdot \mathbf{S}^{(k)} \cdot \mathbf{A}^T\right\|_2}. \tag{6.2}$$

However, as the matrix multiplication may contain $O(n^3)$ operations, it requires $O(kn^3)$ time and $O(n^2)$ space to compute Eq.(6.2) for k iterations, which may be quite expensive.

In this section, we study the optimization techniques to improve the computation of SimFusion+. Our key observation is that SimFusion+ computation can be converted into finding the dominant eigenvector of the UAM $\mathbf{A}$. The idea is to calculate the dominant eigenvector of $\mathbf{A}$ once, offline, for the preprocessing, and then it can be effectively memorized to compute similarity at query time.

We first revisit the definition of the dominant eigenvector.

Definition 6.4 ([Wil07, p.379])**.** The dominant eigenvector of the $\mathbf{X}$ is an eigenvector, denoted by $\sigma_{\max}(\mathbf{X})$, corresponding to the eigenvalue λ of the largest absolute value of $\mathbf{X}$ such that

$$\mathbf{X} \cdot \sigma_{\max}(\mathbf{X}) = \lambda \cdot \sigma_{\max}(\mathbf{X}) \quad \text{with } \|\sigma_{\max}(\mathbf{X})\|_2 = 1. \qquad \square$$

The dominant eigenvector of the UAM can be utilized for speeding up SimFusion+ computation based on the following proposition.

Proposition 6.5. Let $\mathbf{A}$ be the UAM of network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$. The SimFusion+ matrix $\mathbf{S}$ can be computed as

$$[\mathbf{S}]_{i,j} = [\sigma_{\max}(\mathbf{A})]_i \times [\sigma_{\max}(\mathbf{A})]_j, \tag{6.3}$$

where $[\star]_{i,j}$ denotes the (i,j)-entry of a matrix, and $[\star]_i$ denotes the i-th entry of a vector. $\square$

[1] Note that the uniqueness of the SimFusion+ solution guarantees that $\mathbf{S}$ is insensitive to the initial guess $\mathbf{S}^{(0)}$. For convenience, we choose $\mathbf{S}^{(0)} = \mathbf{1}$ of all 1s, which can be interpreted as "initially, no other vertex-pair is presumably more similar than itself".

Proof. We shall use the knowledge of Kronecker product ($\otimes$) and *vec* operator (see [Lau04, p.139] for a detailed description).

(i) We first prove that $vec(\mathbf{S}) = \sigma_{\max}(\mathbf{A} \otimes \mathbf{A})$.

Taking $vec(\star)$ on both sides of Eq.(6.2) and applying Kronecker property $vec(\mathbf{BCD}^T) = (\mathbf{D} \otimes \mathbf{B}) \cdot vec(\mathbf{C})$ [Lau04, p.147] yield

$$vec(\mathbf{S}^{(k+1)}) = \frac{(\mathbf{A} \otimes \mathbf{A}) \cdot vec(\mathbf{S}^{(k)})}{\left\|(\mathbf{A} \otimes \mathbf{A}) \cdot vec(\mathbf{S}^{(k)})\right\|_2}. \tag{6.4}$$

Let $\mathbf{x}^{(k)} \triangleq vec(\mathbf{S}^{(k)})$ and $\mathbf{M} \triangleq \mathbf{A} \otimes \mathbf{A}$. Then Eq.(6.4) having the form $\mathbf{x}^{(k+1)} = \mathbf{M}\mathbf{x}^{(k)}/\|\mathbf{M}\mathbf{x}^{(k)}\|_2$ fits the *power iteration* paradigm [Wil07, pp.381], which follows that the sequence $\{\mathbf{x}^{(k)}\}$ converges to the dominant eigenvector of $\mathbf{M}$. This in turn implies

$$vec(\mathbf{S}) \triangleq \lim_{k\to\infty} vec(\mathbf{S}^{(k)}) = \sigma_{\max}(\mathbf{A} \otimes \mathbf{A}).$$

One caveat is that the convergence of $vec(\mathbf{S}^{(k)})$ is ensured by the positivity of $\mathbf{A} \otimes \mathbf{A}$. [2] This is true because $\mathbf{A}$ is positive and the self-Kronecker product of two positive matrices preserves positivity.

(ii) We next show that $\sigma_{\max}(\mathbf{A} \otimes \mathbf{A}) = \sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A})$.

Since $\mathbf{A} \cdot \sigma_{\max}(\mathbf{A}) = \lambda \cdot \sigma_{\max}(\mathbf{A})$, it follows that

$$\begin{aligned}(\mathbf{A} \otimes \mathbf{A}) \cdot (\sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A})) &= (\mathbf{A}\sigma_{\max}(\mathbf{A})) \otimes (\mathbf{A}\sigma_{\max}(\mathbf{A})) \\ &= (\lambda \cdot \sigma_{\max}(\mathbf{A})) \otimes (\lambda \cdot \sigma_{\max}(\mathbf{A})) = \lambda^2 \cdot (\sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A})).\end{aligned}$$

This implies that the dominant eigenvector of $\mathbf{A}\otimes\mathbf{A}$ is actually the self-Kronecker product of the dominant eigenvector of $\mathbf{A}$. Hence,

$$vec(\mathbf{S}) = \sigma_{\max}(\mathbf{A} \otimes \mathbf{A}) = \sigma_{\max}(\mathbf{A}) \otimes \sigma_{\max}(\mathbf{A}).$$

It can be noticed that the (i, j)-entry of the matrix $\mathbf{S}$ (*i.e.*, the $((i-1) \times n + j)$-th entry of the vector $vec(\mathbf{S})$) is exactly the product of the i-th and j-th entries of $\sigma_{\max}(\mathbf{A})$. Thus, Eq.(6.3) holds. □

[2] According to the Perron-Frobenius theorem [Wil07, p.383], the positivity of $\mathbf{A}\otimes\mathbf{A}$ ensures that there exists a unique dominant eigenvector of $\mathbf{A} \otimes \mathbf{A}$ associated with its eigenvalue being strictly greater in magnitude than its other eigenvalues.

Proposition 6.5 provides the efficient technique for accelerating SimFusion+ computation. The central point in optimizing $\mathbf{S}$ computation is that only *matrix-vector* multiplication is used for computing $\sigma_{\max}(\mathbf{A})$. Once calculated, the vector $\sigma_{\max}(\mathbf{A})$ is memorized and thus will not be recomputed when subsequently required, as opposed to the naive *matrix-matrix* multiplication in Eq.(6.2).

Example 6.6. Consider the graph $\mathcal{G}_1$ in Fig.6.1 with its UAM $\mathbf{A}$ already computed in Example 6.3. The dominant eigenvector of $\mathbf{A}$ is

$$\sigma_{\max}(\mathbf{A}) = [.431\ .673\ .451\ .322\ .232]^T.$$

Then using Eq.(6.3) for computing $\mathbf{S}$ yields

$$\mathbf{S} = \begin{bmatrix} .186 & .290 & .194 & .139 & .100 \\ .290 & .453 & .304 & .217 & .156 \\ .194 & .304 & .203 & .145 & .105 \\ .139 & .217 & .145 & .104 & .075 \\ .100 & .156 & .105 & .075 & .054 \end{bmatrix}.$$

Note that $\sigma_{\max}(\mathbf{A})$ is calculated only once for the preprocessing and can be used for computing any entry of $\mathbf{S}$ at query time, *e.g.*,

$$[\mathbf{S}]_{1,2} = [\sigma_{\max}(\mathbf{A})]_1 \times [\sigma_{\max}(\mathbf{A})]_2 = .431 \times .673 = .290.$$
$$[\mathbf{S}]_{1,3} = [\sigma_{\max}(\mathbf{A})]_1 \times [\sigma_{\max}(\mathbf{A})]_3 = .431 \times .451 = .194. \quad \square$$

Regarding computational complexity, our approach only needs $O(km)$ preprocessing time and $O(n)$ space to compute $\sigma_{\max}(\mathbf{A})$ by using the following *power iteration* [Wil07, pp.381]:

$$\boldsymbol{\xi}^{(0)} = \mathbf{e}, \quad \boldsymbol{\xi}^{(k+1)} = \frac{\mathbf{A}\boldsymbol{\xi}^{(k)}}{\|\mathbf{A}\boldsymbol{\xi}^{(k)}\|_2} = \frac{\tilde{\mathbf{A}}\boldsymbol{\xi}^{(k)} + \gamma^{(k)}\mathbf{e}}{\|\tilde{\mathbf{A}}\boldsymbol{\xi}^{(k)} + \gamma^{(k)}\mathbf{e}\|_2}, \tag{6.5}$$

where $\mathbf{e} \triangleq (1, \cdots, 1)^T \in \mathbb{R}^n$ and $\gamma^{(k)} \triangleq \frac{1}{n^2}\sum_{i=1}^{n}[\boldsymbol{\xi}^{(k)}]_i$. [3] The existence and uniqueness of the dominant eigenvector $\sigma_{\max}(\mathbf{A})$ is guaranteed by the combination of Perron-Frobenius

[3] The correctness of Eq.(6.5) can be proved as follows:

$$\mathbf{A}\boldsymbol{\xi}^{(k)} = (\tilde{\mathbf{A}} + \tfrac{1}{n^2}\mathbf{e}\mathbf{e}^T)\boldsymbol{\xi}^{(k)} = \tilde{\mathbf{A}}\boldsymbol{\xi}^{(k)} + \gamma^{(k)}\mathbf{e} \text{ with } \gamma^{(k)} = \tfrac{1}{n^2}\mathbf{e}^T\boldsymbol{\xi}^{(k)}.$$

theorem [Wil07, p.383] and the positivity of $\mathbf{A}$. Thus, by applying the power method, the sequence $\{\boldsymbol{\xi}^{(k)}\}$ converges to $\sigma_{\max}(\mathbf{A})$. Then, with $\sigma_{\max}(\mathbf{A})$ being memorized, only $O(1)$ time is required at query stage for computing each entry of $\mathbf{S}$ via Eq.(6.3). Indeed, due to $\mathbf{S}$ symmetry, only $n(n+1)/2$ entries $[\mathbf{S}]_{i,j}$ $(i \leq j)$ need to be computed. In contrast to the $O(kn^3)$ time and $O(n^2)$ space of the conventional iterations, our approach is a significant improvement achieved by $\sigma_{\max}(\mathbf{A})$ computation.

Our method of memorizing $\sigma_{\max}(\mathbf{A})$ can extra accelerate SimFusion+ computation when only a small portion of similarity values of $\mathbf{S}$ need to be computed. Specifically, for certain applications like K-nearest neighbor (KNN) queries, given a vertex i as a query, one needs to retrieve the top-K ($\ll n$) most similar vertices in a graph by computing the i-th row of $\mathbf{S}$. Before Proposition 6.5 is introduced, computing the similarity of only one vertex-pair still requires $O(kn^3)$ time. In contrast, using the memorized $\sigma_{\max}(\mathbf{A})$, we only need $O(1)$ time for computing a single entry of $\mathbf{S}$ at query time. In fact, for KNN queries, after $\sigma_{\max}(\mathbf{A})$ is memorized with its entries sorted in an descending order for the preprocessing, it only takes constant time to retrieve the top-K results at query stage.

Proposition 6.5 also gives an interesting characterization of the SimFusion+ matrix.

Corollary 6.7. The SimFusion+ matrix $\mathbf{S}$ is a rank 1 matrix, that is, for any two rows of $\mathbf{S}$, say $[\mathbf{S}]_{x,*}$ and $[\mathbf{S}]_{y,*}$, there exists a scalar ω such that $[\mathbf{S}]_{x,*} = \omega \times [\mathbf{S}]_{y,*}$. □

Proof. Applying Eq.(6.3) to $\mathbf{S}$, we obtain that for any two rows of $\mathbf{S}$, there exists $\omega = [\sigma_{\max}(\mathbf{A})]_x / [\sigma_{\max}(\mathbf{A})]_y$ *s.t.* $[\mathbf{S}]_{x,*} = \omega \times [\mathbf{S}]_{y,*}$. This implies that any row of $\mathbf{S}$ can be expressed in terms of any other row of $\mathbf{S}$. Hence, the rank of $\mathbf{S}$ is 1. □

6.4 Estimating SimFusion+ With Better Accuracy

After the dominant eigenvector $\sigma_{\max}(\mathbf{A})$ has been suggested to speed up SimFusion+ computation, the algorithm presented in this section can guarantee more accurate similarity results.

The main idea of our approach is to leverage an orthogonal subspace for "upper-triangularizing" the UAM $\mathbf{A}$ ($n \times n$ dimension) into a *small* matrix $\mathbf{T}_k$ ($k \times k$ dimension)

with $k \ll n$. Due to $\mathbf{T}_k$ small size and almost "upper-triangularity", computing the dominant eigenvector $\sigma_{\max}(\mathbf{T}_k)$ is far less costly than straightforwardly computing $\sigma_{\max}(\mathbf{A})$. We show that the choice of k provides a user-controlled accuracy over the similarity scores. The underlying rationale is that the dominant eigenvector of a matrix can be well-preserved by an orthogonal transformation.

We first use the technique of the Arnoldi decomposition [Saa03] to build an order-k orthogonal subspace for the UAM $\mathbf{A}$.

Lemma 6.8 ([Saa03, pp.25-33])**.** Let $\mathbf{A}$ be an $n \times n$ matrix. Then, for every $k = 1, 2, \cdots$, we have the following results.

(a) There exists a unique $k \times k$ almost triangular matrix $\mathbf{T}_k$ *s.t.*

$$\mathbf{V}_k^T \mathbf{A} \mathbf{V}_k = \mathbf{T}_k, \tag{6.6}$$

where $\mathbf{V}_k = [\mathbf{v}_1 \,|\, \mathbf{v}_2 \,|\, \cdots |\, \mathbf{v}_k]$ is an $n \times k$ matrix consisting of k orthonormal column-vectors $\mathbf{v}_i \in \mathbb{R}^n \;\; (i = 1, \cdots, k)$.

(b) The difference between $\mathbf{A}\mathbf{V}_k$ and $\mathbf{V}_k\mathbf{T}_k$ is a zero matrix except the last column. Precisely, there exist a small scalar δ_k and an orthonormal vector $\mathbf{v}_{k+1} \in \mathbb{R}^n$ such that

$$\mathbf{A}\mathbf{V}_k - \mathbf{V}_k\mathbf{T}_k = \delta_k \mathbf{v}_{k+1} \mathbf{e}_k^T, \tag{6.7}$$

where $\mathbf{e}_k = (0, \cdots, 0, 1)^T \in \mathbb{R}^k$ is a unit vector. □

As depicted in Fig.6.3, by using the upper-triangularization process [4], the matrix $\mathbf{A} \in \mathbb{R}^{n \times n}$ can be transformed into the small almost triangular $\mathbf{T}_k \in \mathbb{R}^{k \times k}$ by the $n \times k$ orthonormal matrix $\mathbf{V}_k$ for every iteration. As k increases, $\mathbf{T}_{k+1}$ can be iteratively obtained by bordering the matrix $\mathbf{T}_k$ at the last iteration (*i.e.,* $\mathbf{T}_{k+1} = \left[\begin{smallmatrix} \mathbf{T}_k & \star \\ \star & \star \end{smallmatrix}\right]$), and $\mathbf{V}_{k+1}$ by augmenting the matrix $\mathbf{V}_k$ at the last iteration with the vector $\mathbf{v}_{k+1}$ (*i.e.,* $\mathbf{V}_{k+1} = [\mathbf{V}_k \,|\, \mathbf{v}_{k+1}]$). When $k = rank(\mathbf{A})$, it follows that $\delta_k = 0$.

Example 6.9. Consider the network $\mathcal{G}_1$ in Fig.6.1 and its UAM $\mathbf{A} = \tilde{\mathbf{A}} + \mathbf{1}/5^2$ in Example 6.3. For $k = 3$, $\exists\ \mathbf{V}_3 = [\mathbf{v}_1 \,|\, \mathbf{v}_2 \,|\, \mathbf{v}_3] \in \mathbb{R}^{5 \times 3}$ mapping $\mathbf{A} \in \mathbb{R}^{5 \times 5}$ into $\mathbf{T}_3 \in \mathbb{R}^{3 \times 3}$

[4] From the computational viewpoint, $\mathbf{V}_k, \mathbf{T}_k, \mathbf{v}_k$ and δ_k in Eq.(6.7) can be obtained by an algorithm in our later developments.

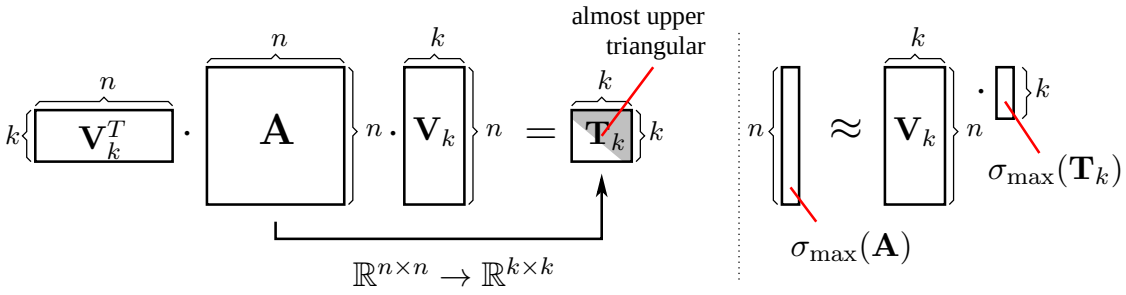


Figure 6.3: Upper Triangular Process of UAM

s.t. $\mathbf{T}_3 = \mathbf{V}_3^T \mathbf{A} \mathbf{V}_3$, where

$$\tilde{\mathbf{A}} = \begin{bmatrix} \frac{1}{2} & \frac{1}{6} & \frac{1}{6} & \frac{1}{3} & 0 \\ \frac{1}{6} & \frac{7}{12} & \frac{7}{12} & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{6} & \frac{7}{12} & 0 & \frac{1}{8} & \frac{1}{8} \\ \frac{1}{3} & \frac{1}{8} & \frac{1}{8} & 0 & \frac{5}{12} \\ 0 & \frac{1}{8} & \frac{1}{8} & \frac{5}{12} & 0 \end{bmatrix}, \ \mathbf{T}_3 = \begin{bmatrix} 1.08 & .298 & 0 \\ .298 & .190 & .359 \\ 0 & .359 & -.083 \end{bmatrix}, \ \mathbf{V}_3 = \left[\begin{array}{c|c|c} .447 & .125 & -.089 \\ .447 & .750 & .044 \\ .447 & -.125 & .710 \\ .447 & -.125 & -.696 \\ .447 & -.625 & .032 \end{array}\right].$$

$\exists\ \delta_3 = .231, \mathbf{v}_4 = [-.881\ .328\ .137\ .280\ .135]^T$ *s.t.* Eq.(6.7) holds.

(Please refer to Example 6.12 for a detailed iterative process) □

In light of Lemma 6.8, we next provide an error estimate for SimFusion+ similarity when using $\sigma_{\max}(\mathbf{T}_k)$ to compute $\sigma_{\max}(\mathbf{A})$.

Error Estimation. We define a k-approximation similarity matrix $\hat{\mathbf{S}}_k$ over a low-order parameter k:

$$[\hat{\mathbf{S}}_k]_{i,j} = [\mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k)]_i \times [\mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k)]_j, \tag{6.8}$$

where $\mathbf{V}_k$ and $\mathbf{T}_k$ can be obtained from Lemma 6.8.

To differentiate $\hat{\mathbf{S}}_k$ from $\mathbf{S}$, we shall refer to $\mathbf{S}$ as *exact* similarity.

The following estimate for the approximate similarity $\hat{\mathbf{S}}_k$ with respect to the exact $\mathbf{S}$ can be established.

Proposition 6.10. For every $k = 1, 2, \cdots$, the following estimate holds:

$$\|\hat{\mathbf{S}}_k - \mathbf{S}\|_2 \leq \epsilon_k, \tag{6.9}$$

where

$$\epsilon_k = 2 \times |\delta_k \times [\sigma_{\max}(\mathbf{T}_k)]_k|, \tag{6.10}$$

and δ_k is a small scalar given in Eq.(6.7); $[\sigma_{\max}(\mathbf{T}_k)]_k$ is the k-th entry of the dominant eigenvector of $\mathbf{T}_k$. □

Proof. Let $\boldsymbol{\psi}(\mathbf{x}) = \mathbf{A}\mathbf{x} - \alpha(\mathbf{x}) \cdot \mathbf{x}$ be a vector function of $\mathbf{x} \in \mathbb{R}^n$, with $\alpha(\mathbf{x})$ being a real function of $\mathbf{x}$. To simplify notations, we shall denote by α_k the dominant eigenvalue of $\mathbf{T}_k$, and

$$\boldsymbol{\eta}_k = \sigma_{\max}(\mathbf{T}_k), \quad \boldsymbol{\xi}_k = \mathbf{V}_k \cdot \sigma_{\max}(\mathbf{T}_k), \quad \boldsymbol{\xi} = \sigma_{\max}(\mathbf{A}).$$

Using $\mathbf{T}_k\boldsymbol{\eta}_k = \alpha_k\boldsymbol{\eta}_k$ and Eq.(6.7) in Lemma 6.8, we have

$$\begin{aligned}\boldsymbol{\psi}(\boldsymbol{\xi}_k) &= \mathbf{V}_k\mathbf{T}_k\boldsymbol{\eta}_k + \delta_k\mathbf{v}_{k+1}\mathbf{e}_k^T\boldsymbol{\eta}_k - \alpha_k\mathbf{V}_k\boldsymbol{\eta}_k \\ &= \delta_k\mathbf{v}_{k+1}(\mathbf{e}_k^T\boldsymbol{\eta}_k) = \delta_k[\boldsymbol{\eta}_k]_k\mathbf{v}_{k+1},\end{aligned}$$

where $[\boldsymbol{\eta}_k]_k$ denotes the k-th entry of $\boldsymbol{\eta}_k$. Hence,

$$\|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 \le \|\boldsymbol{\psi}(\boldsymbol{\xi}_k)\|_2 = \left|\delta_k[\boldsymbol{\eta}_k]_k\right| \|\mathbf{v}_{k+1}\|_2 = \left|\delta_k[\boldsymbol{\eta}_k]_k\right|.$$

Since $vec(\mathbf{S}_k) = \boldsymbol{\xi}_k \otimes \boldsymbol{\xi}_k$ and $vec(\mathbf{S}) = \boldsymbol{\xi} \otimes \boldsymbol{\xi}$, we have

$$\begin{aligned}\|\mathbf{S}_k - \mathbf{S}\|_2 &= \|vec(\mathbf{S}_k) - vec(\mathbf{S})\|_2 = \|\boldsymbol{\xi}_k \otimes \boldsymbol{\xi}_k - \boldsymbol{\xi} \otimes \boldsymbol{\xi}\|_2 \\ &= \|\boldsymbol{\xi}_k \otimes (\boldsymbol{\xi}_k - \boldsymbol{\xi}) + (\boldsymbol{\xi}_k - \boldsymbol{\xi}) \otimes \boldsymbol{\xi}\|_2 \\ &\le \underbrace{\|\boldsymbol{\xi}_k\|_2}_{\le 1} \cdot \|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 + \|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 \cdot \underbrace{\|\boldsymbol{\xi}\|_2}_{\le 1} \\ &= 2 \times \|\boldsymbol{\xi}_k - \boldsymbol{\xi}\|_2 \le 2 \times |\delta_k \times [\boldsymbol{\eta}_k]_k|,\end{aligned}$$

which completes the proof. □

The parameter ϵ_k is intended as a user control over the difference between the approximate and the exact similarity matrices, and hence ϵ_k is generally chosen by a user. Provided that k is selected to satisfy Eq.(6.10), Proposition 6.10 states that the gap between the approximate and the exact similarity scores does not exceed ϵ_k.

Example 6.11. Consider the network $\mathcal{G}_1$ in Fig.6.1 and the matrix $\mathbf{T}_3, \mathbf{V}_3, \delta_3$ given in Example 6.9. For $k = 3$, we have

$$\sigma_{\max}(\mathbf{T}_3) = [.945 \ .316 \ .089]^T.$$

Therefore, $\mathbf{V}_3 \cdot \sigma_{\max}(\mathbf{T}_3) = [.454\ .663\ .447\ .321\ .228]^T$. Then applying Eq.(6.8) and the exact $\mathbf{S}$ in Example 6.6 yields

$$\hat{\mathbf{S}}_3 = \{\text{Using Eq.(6.8)}\} = \begin{bmatrix} .206 & .301 & .203 & .146 & .103 \\ .301 & .440 & .296 & .213 & .151 \\ .203 & .296 & .199 & .143 & .102 \\ .146 & .213 & .143 & .102 & .073 \\ .103 & .151 & .102 & .073 & .051 \end{bmatrix}$$

$$\hat{\mathbf{S}}_3 - \mathbf{S} = \{\text{Using } \mathbf{S} \text{ in Example 6.6}\} = .01 \times \begin{bmatrix} 2.02 & 1.09 & .83 & .67 & .34 \\ 1.09 & -1.33 & -.75 & -.41 & -.51 \\ .83 & -.75 & -.40 & -.21 & -.30 \\ .67 & -.41 & -.21 & -.09 & -.17 \\ .34 & -.51 & -.30 & -.17 & -.20 \end{bmatrix}$$

We note that the gap between $\mathbf{S}$ and $\hat{\mathbf{S}}_k$ for $k = 3$ is actually

$$\|\hat{\mathbf{S}}_3 - \mathbf{S}\|_2 = .0257,$$

which is smaller than ϵ_k (using Eq.(6.10) with $\delta_3 = .231$)

$$\epsilon_k = 2 \times |.231 \times .089| = .0411. \qquad \square$$

Notice that if $k = rank(\mathbf{A})$ ($\ll n$), [5] then $\epsilon_k = 0$ and the k-approximation similarity matrix $\hat{\mathbf{S}}_k$ becomes the conventional exact USM $\mathbf{S}$. From this perspective, the k-approximation similarity can be regarded as a generalization for the conventional similarity.

One of the possible ways of choosing an appropriate low order k for achieving the desired accuracy ϵ is to calculate the estimation error ϵ_k from Eq.(6.10) in an a-posteriori fashion after each iteration $k = 1, 2, \cdots$. [6] The iterative process stops once $\epsilon_k \leq \epsilon$. Due to ϵ_k decreasing monotonicity, such k is the minimum low order *s.t.* $\|\hat{\mathbf{S}}_k - \mathbf{S}\|_2 \leq \epsilon$. More concretely, the residual δ_k in Eq.(6.7) (Lemma 6.8) approaches 0 as k is increased to n, which implies

$$\epsilon_k = 2 \times |\delta_k| \times |[\sigma_{\max}(\mathbf{T}_k)]_k| \leq 2 \times |\delta_k|.$$

Hence, the condition $\epsilon_k \leq \epsilon$ (with ϵ_k being obtained from Eq.(6.10)) can be used as a stopping criterion for determining the minimum low order k needed for the desired accuracy ϵ.

[5] When k is set to $\text{argmin}_k\{\delta_k = 0\}$ (which is practically much smaller than $rank(\mathbf{A})$), $\epsilon_k = \epsilon_{k+1} = \cdots = \epsilon_n = 0$.

[6] As increased by 1 per iteration, the low order parameter k equals the iteration number.

Capitalizing on Eq.(6.8) and Proposition 6.10, below we provide an algorithm for SimFusion+ computation with accuracy guarantee.

Algorithm. SimFusion+ is shown in Algorithm 6.1. It takes as input a network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, a desired accuracy ϵ, and a vertex pair (u, v); it returns the approximate similarity $\hat{s}(u, v)$ such that $|\hat{s}(u, v) - s(u, v)| \leq \epsilon$ with $s(u, v)$ being the exact value.

Before illustrating the algorithm, we first present the notations it uses. (a) $[\mathbf{T}_k]_{i,j}$ is the (i, j)-entry of the matrix $\mathbf{T}_k$, and $[\sigma_{\max}(\mathbf{T}_k)]_i$ is the i-th entry of the eigenvector $\sigma_{\max}(\mathbf{T}_k)$. (b) $span\{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$ is the set of all linear combinations of vectors $\mathbf{v}_1, \cdots, \mathbf{v}_k$. (c) $\hat{\sigma}_{\max}(\mathbf{A})$ denotes the approximation of $\sigma_{\max}(\mathbf{A})$.

The algorithm SimFusion+ works as follows. It first computes $\mathbf{A}$ and initializes $\mathbf{v}_1$ (lines 1-2). Using $\mathbf{A}$, it then computes $\mathbf{T}_k$ (lines 4-6), δ_k (lines 7-8) and $\mathbf{v}_{k+1}$ (line 12) by orthonormalizing the vector $\mathbf{A}\mathbf{v}_k$ with respect to $\mathbf{v}_1, \cdots, \mathbf{v}_k$ for every iteration; SimFusion+ also calculates $\sigma_{\max}(\mathbf{T}_k)$ (line 9), and utilizes δ_k and $\sigma_{\max}(\mathbf{T}_k)$ to estimate the error ϵ_k (line 10). The process (lines 3-13) iterates until $\epsilon_k \leq \epsilon$, *i.e.,* the minimum low order k is found *s.t.* ϵ_k meets the desired accuracy ϵ (line 11). For such k, the matrix-vector product $[\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_k] \cdot \sigma_{\max}(\mathbf{T}_k)$ is used to approximate the dominant eigenvector of $\mathbf{A}$, and is memorized to compute $\hat{\sigma}_{\max}(\mathbf{A})$ (line 15). The product of the u-th and v-th entries of $\hat{\sigma}_{\max}(\mathbf{A})$ is collected in $\hat{s}(u, v)$, which is returned as the estimated similarity between vertex u and v (lines 17-18).

Example 6.12. We show how SimFusion+ estimates the similarity in $\mathcal{G}_1$ of Example 6.1. Given the desired accuracy $\epsilon = 0.05$, SimFusion+ first initializes the UAM $\mathbf{A}$ (in Example 6.3). It then iteratively computes $\mathbf{T}_k, \mathbf{v}_{k+1}, \delta_k, \sigma_{\max}(\mathbf{T}_k)$ and ϵ_k as follows:

Algorithm 6.1: SimFusion+ $(\mathcal{G}, \epsilon, (u, v))$

Input : Network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, accuracy ϵ, vertex pair (u, v).

Output: Similarity score $s(u, v)$.

compute the matrix $\tilde{\mathbf{A}} \in \mathbb{R}^{n \times n}$ of the UAM $\mathbf{A}$ in $\mathcal{G}$;

initialize $\mathbf{e} \leftarrow (1, 1, \cdots, 1)^T \in \mathbb{R}^n$, and $\mathbf{v}_1 \leftarrow \frac{1}{\sqrt{n}}\mathbf{e}$;

foreach *iteration* $k = 1, 2, \cdots$ **do**

  initialize the auxiliary vector $\mathbf{w} \leftarrow \tilde{\mathbf{A}}\mathbf{v}_k + \frac{1}{n^2}(\mathbf{e}^T\mathbf{v}_k)\mathbf{e}$;

  for $i = 1, 2, \cdots, k-1$ **do**

    compute the almost upper triangular matrix $\mathbf{T}_k \in \mathbb{R}^{k \times k}$:

    $[\mathbf{T}_k]_{i,k-1} \leftarrow \mathbf{v}_k^T \cdot \mathbf{w}$;

    orthogonalize $\mathbf{w}$ *s.t.* $\mathbf{w} \perp span\{\mathbf{v}_1, \cdots, \mathbf{v}_k\}$:

    $\mathbf{w} \leftarrow \mathbf{w} - [\mathbf{T}_k]_{i,k-1} \cdot \mathbf{v}_{k-1}$;

  compute the residual scalar $\delta_k \leftarrow \|\mathbf{w}\|_2$;

  find the dominant eigenvector $\sigma_{\max}(\mathbf{T}_k)$;

  estimate the error $\epsilon_k \leftarrow 2 \times |\delta_k \times [\sigma_{\max}(\mathbf{T}_k)]_k|$;

  if $\epsilon_k \leq \epsilon$ **then** exit for ;

  compute the residual vector $\mathbf{v}_{k+1}$:

  $\mathbf{w} \leftarrow \mathbf{w}/\delta_k$ and $\mathbf{v}_{k+1} \leftarrow \mathbf{w}$;

  free $\delta_k, \sigma_{\max}(\mathbf{T}_k)$;

free $\mathbf{w}, \mathbf{T}_k, \mathbf{v}_{k+1}, \delta_k$;

compute the approximate dominant eigenvector $\hat{\sigma}_{\max}(\mathbf{A})$

$\hat{\sigma}_{\max}(\mathbf{A}) \leftarrow [\mathbf{v}_1|\mathbf{v}_2|\cdots|\mathbf{v}_k] \cdot \sigma_{\max}(\mathbf{T}_k)$;

free $\mathbf{v}_1, \cdots, \mathbf{v}_k, \sigma_{\max}(\mathbf{T}_k)$;

compute the approximate similarity score of (u, v)

$\hat{s}(u, v) \leftarrow [\hat{\sigma}_{\max}(\mathbf{A})]_u \times [\hat{\sigma}_{\max}(\mathbf{A})]_v$;

return $\hat{s}(u, v)$;

#-line	time	memory	operation
4	$O(m)$	$O(n)$	sparse matrix-vector multiplication
6	$O(n)$	$O(n)$	vector dot product
7	$O(n)$	$O(n)$	vector addition and scalar multiplication
8	$O(n)$	$O(n)$	computing the 2-norm of a vector
9	$O(k)$	$O(k)$	using the power iteration
10	$O(1)$	$O(k)$	getting the vector component
12	$O(n)$	$O(n)$	scaling the vector

Table 6.1: Running Time & Memory Space Required per Iteration for Algorithm SimFusion+ in Lines 4-12

k	$\mathbf{T}_k$	$\mathbf{v}_{k+1}$	δ_k	$\sigma_{\max}(\mathbf{T}_k)$	ϵ_k
0	–	$[.447\ .447\ .447\ .447\ .447]^T$	–	–	–
1	[1.08]	$[.125\ .750\ -.125\ -.125\ -.625]^T$	.298	[1]	.596
2	$\begin{bmatrix} 1.08 & .298 \\ .298 & .190 \end{bmatrix}$	$[-.089\ .044\ .710\ -.697\ .032]^T$	.359	$\begin{bmatrix} .957 \\ .290 \end{bmatrix}$	.208
3	$\begin{bmatrix} 1.08 & .298 & 0 \\ .298 & .190 & .359 \\ 0 & .359 & -.083 \end{bmatrix}$	$[-.881\ .329\ .137\ .280\ .135\ .231]^T$	.231	$\begin{bmatrix} .945 \\ .316 \\ .090 \end{bmatrix}$	.041

The iteration terminates at $k = 3$ because the estimation error $\epsilon_3 = .041 \leq \epsilon\ (= .05)$. SimFusion+ then memorizes $\hat{\sigma}_{\max}(\mathbf{A}) = [\mathbf{v}_1|\mathbf{v}_2|\mathbf{v}_3] \cdot \sigma_{\max}(\mathbf{T}_3)$ and returns the similarity $\hat{s}(u, v)$, *i.e.*, the (u, v) entry of $\hat{\mathbf{S}}_3$, as shown in Example 6.11. □

We next analyze the time and space complexity of SimFusion+ .

Running Time. The algorithm consists of two phases: preprocessing (lines 1-16), and on-line query (lines 17-18).

(i) For the preprocessing, (a) it takes $O(m)$ time to compute $\tilde{\mathbf{A}}$ (line 1) and $O(n)$ time to initialize $\mathbf{v}_1$ (line 2). (b) The total time of the **for** loop is analyzed in Table 6.1 (line 3-13), which is bounded by $O(m + 4n + k + 1)$ for each iteration. (c) It takes $O(kn)$ time to compute $\hat{\sigma}_{\max}(\mathbf{A})$ (line 15). Hence, the total time in this phase is $O(m + k(m + 4n + k + 1) + kn)$, which is bounded by $O(km)$.

(ii) The on-line query phase (lines 17-18) can be done in constant time for each query by virtue of $\hat{\sigma}_{\max}(\mathbf{A})$ memorization.

Combining (i) and (ii), the query time of SimFusion+ is in $O(1)$, plus an $O(km)$-time precomputation.

Memory Space. (i) In the precomputation, (a) initializing $\tilde{\mathbf{A}}$ and $\mathbf{v}_1$ takes $O(n)$ space (lines 1-2). (b) For each iteration k, the space complexity is analyzed in Table 6.1, which is bounded by $O(n)$ (line 3-13). (c) As the **for** loop terminates, only $\mathbf{v}_1, \cdots, \mathbf{v}_k$ and $\sigma_{\max}(\mathbf{T}_k)$ are kept in memory, yielding $O(kn + k)$ space; the other intermediate results can be freed (line 14). (d) Computing $\hat{\sigma}_{\max}(\mathbf{A})$ takes $O(k)$ space (line 15). Once computed, $\hat{\sigma}_{\max}(\mathbf{A})$ is memorized, yielding $O(n)$ space; $\mathbf{v}_1, \cdots, \mathbf{v}_k$ and $\sigma_{\max}(\mathbf{T}_k)$ are not used subsequently and thus can be freed (line 16).

(ii) For the on-line query (lines 17-18), $\hat{s}(u, v)$ can be computed in $O(n)$ space with $\hat{\sigma}_{\max}(\mathbf{A})$ memorized.

Taking (i) and (ii) together, the total space is bounded by $O(kn)$.

6.5 Incremental SimFusion+

For certain applications like social networks, graphs are frequently modified [LHH+10]. It is too costly to recalculate similarities every time when edges in the graphs are updated. This motivates us to study the following *incremental SimFusion+ estimating problem.*

Given a network $\mathcal{G}$, the eigen-information in $\mathcal{G}$, and a list $\bar{\mathcal{G}}$ of updates (edge deletions and insertions) to $\mathcal{G}$, it is to compute the new USM $\mathbf{S}'$ in $\mathcal{G}'$. Here $\mathcal{G}'$ is the updated $\mathcal{G}$, denoted by $\mathcal{G} + \bar{\mathcal{G}}$.

The idea is to maximally reuse the eigen-information in $\mathcal{G}$ when computing $\mathbf{S}'$. The observation is that $\bar{\mathcal{G}}$ is often small in practice; hence, $\mathbf{S}'(= \mathbf{S} + \bar{\mathbf{S}})$ is slightly different from $\mathbf{S}$. It is far less costly to find the change $\bar{\mathbf{S}}$ to the old $\mathbf{S}$ than to recalculate the new $\mathbf{S}'$ from scratch. The main result in this section is the following.

Theorem 6.13. *The incremental SimFusion+ estimating problem is solvable in $O(\delta n)$ time and $O(n)$ space for every vertex pair, where δ is the number of edges affected by the*

update $\bar{\mathcal{G}}$. □

As we shall see later, δ captures the size of areas in a graph $\mathcal{G}$ that is affected by updates $\bar{\mathcal{G}}$; hence δ is much smaller than n when $\bar{\mathcal{G}}$ is small. That is, the incremental SimFusion+ can be performed more efficiently than computing similarities in $\mathcal{G}'$. This suggests that we compute the eigenvector of $\mathbf{A}$ in $\mathcal{G}$ once, and then incrementally compute SimFusion+ when $\mathcal{G}$ is updated.

To prove Theorem 6.13, we first introduce a notion of incremental UAM. We then devise an incremental algorithm for handling batch edge updates with the desired bound.

6.5.1 Incremental Unified Adjacency Matrix

Consider an old network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$ and a new $\mathcal{G}' = (\mathcal{D}, \mathcal{R}')$.

Incremental UAM. The matrix $\bar{\mathbf{A}}$ is said to be the *incremental UAM* of the update $\bar{\mathcal{G}}$ ($= \mathcal{G}' - \mathcal{G}$, *i.e.,* a list of edge insertions and deletions) *iff* $\bar{\mathbf{A}} = \mathbf{A}' - \mathbf{A}$, where $\mathbf{A}$ and $\mathbf{A}'$ are the UAMs of the old network $\mathcal{G}$ and the new $\mathcal{G}'$, respectively.

Intuitively, the nonzero entries of $\bar{\mathbf{A}}$ can identify the edges in $\mathcal{G}$ that is affected by updates $\bar{\mathcal{G}}$. Typically, $\bar{\mathbf{A}}$ is a sparse matrix when δ is small. Indeed, the number of nonzero entries in $\bar{\mathbf{A}}$ is bounded by $O(\delta n)$, which represents the costs that are inherent to the incremental problem itself, *i.e.,* the amount of work absolutely necessary to be performed for the problem.

Using $\bar{\mathbf{A}}$, we next provide a strategy for incrementally computing SimFusion+ similarity.

Proposition 6.14. Given a network $\mathcal{G}$ and an update $\bar{\mathcal{G}}$ to $\mathcal{G}$, let $\mathbf{A}$ be the UAM of $\mathcal{G}$, and $\bar{\mathbf{A}}$ the incremental UAM of $\bar{\mathcal{G}}$. Then the new USM $\mathbf{S}'$ of the new network $\mathcal{G}'$ ($= \mathcal{G} + \bar{\mathcal{G}}$) can be computed as

$$[\mathbf{S}']_{i,j} = [\boldsymbol{\xi}']_i \cdot [\boldsymbol{\xi}']_j \text{ with } [\boldsymbol{\xi}']_i = [\boldsymbol{\xi}_1]_i + \textstyle\sum_{p=2}^{n} c_p \times [\boldsymbol{\xi}_p]_i$$
$$c_p = \frac{\boldsymbol{\xi}_p^T \cdot \boldsymbol{\eta}}{\alpha_p - \alpha_1} \text{ and } \boldsymbol{\eta} = \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1. \tag{6.11}$$

where $\boldsymbol{\xi}_p$ is the eigenvector of $\mathbf{A}$ corresponding to the eigenvalue α_p with $\|\boldsymbol{\xi}_p\|_2 = 1$, and $\boldsymbol{\xi}_1$ is the dominant eigenvector of $\mathbf{A}$. □

Proof. For the new graph $\mathcal{G}'$, let $\boldsymbol{\xi}'$ be the dominant eigenvector of $\mathbf{A}'$ with its eigenvalue α', and $\bar{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}' - \boldsymbol{\xi}_1$, $\bar{\alpha}_1 = \alpha' - \alpha_1$, $\bar{\mathbf{A}} = \mathbf{A}' - \mathbf{A}$. Then, $\mathbf{A}'\boldsymbol{\xi}' = \alpha'\boldsymbol{\xi}'$ can be rewritten as

$$(\mathbf{A} + \bar{\mathbf{A}})(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1) = (\alpha_1 + \bar{\alpha})(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1).$$

Expanding the above equation, eliminating $\bar{\mathbf{A}}\bar{\boldsymbol{\xi}}_1$ and $\bar{\alpha}_1\bar{\boldsymbol{\xi}}_1$ (the high-order infinitesimals of $\bar{\boldsymbol{\xi}}_1$), and using $\mathbf{A}\boldsymbol{\xi}_1 = \alpha_1\boldsymbol{\xi}_1$, we have

$$\mathbf{A}\bar{\boldsymbol{\xi}}_1 + \boldsymbol{\eta} = \alpha_1\bar{\boldsymbol{\xi}}_1 + \bar{\alpha}_1\boldsymbol{\xi}_1 \text{ with } \boldsymbol{\eta} = \bar{\mathbf{A}}\boldsymbol{\xi}_1. \tag{6.12}$$

Since $\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_n$ of $\mathbf{A}$ constitute a basis for $\mathbb{R}^n$, there exist scalars $c_1, \cdots, c_n$ *s.t.* $\bar{\boldsymbol{\xi}}_1 = c_1\boldsymbol{\xi}_1 + \cdots + c_n\boldsymbol{\xi}_n$. Substituting this into Eq.(6.12) and pre-multiplying both sides by $\boldsymbol{\xi}_j^T$ produce

$$\boldsymbol{\xi}_j^T \sum_{i=1}^{n} c_i\alpha_i\boldsymbol{\xi}_i + \boldsymbol{\xi}_j^T\boldsymbol{\eta} = \alpha_1\boldsymbol{\xi}_j^T \sum_{i=1}^{n} c_i\boldsymbol{\xi}_i + \bar{\alpha}_1\boldsymbol{\xi}_j^T\boldsymbol{\xi}_1. \quad (j = 2, \cdots, n)$$

Due to $\boldsymbol{\xi}_i$ orthonormality $(i = 1, \cdots, n)$, we have

$$\underbrace{c_j\alpha_j\boldsymbol{\xi}_j^T\boldsymbol{\xi}_j}_{=c_j\alpha_j} + \boldsymbol{\xi}_j^T\boldsymbol{\eta} = \underbrace{c_j\alpha_1\boldsymbol{\xi}_j^T\boldsymbol{\xi}_j}_{=c_j\alpha_1} + \underbrace{\bar{\alpha}_1\boldsymbol{\xi}_j^T\boldsymbol{\xi}_1}_{=0}.$$

Hence, $c_j = (\boldsymbol{\xi}_j^T\boldsymbol{\eta})/(\alpha_j - \alpha_1)$ with $\boldsymbol{\eta} = \bar{\mathbf{A}}\boldsymbol{\xi}_1 \quad (j = 2, \cdots, n)$.

To determine c_1, we use the identity $(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1)^T(\boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1) = 1$. Expanding the left-hand side, eliminating the high-order infinitesimal $\bar{\boldsymbol{\xi}}_1^T\bar{\boldsymbol{\xi}}_1$, and replacing $\bar{\boldsymbol{\xi}}_1$ with $c_1\boldsymbol{\xi}_1 + \cdots + c_n\boldsymbol{\xi}_n$, we have

$$\underbrace{\boldsymbol{\xi}_1^T\boldsymbol{\xi}_1}_{=1} + \underbrace{\boldsymbol{\xi}_1^T \sum_{i=1}^{n} c_i\boldsymbol{\xi}_i}_{=c_1} + \underbrace{\sum_{i=1}^{n} c_i\boldsymbol{\xi}_i^T\boldsymbol{\xi}_1}_{=c_1} = 1.$$

Due to $\boldsymbol{\xi}_1, \cdots, \boldsymbol{\xi}_n$ orthonormality, it follows that $c_1 = 0$. Thus,

$$vec(\mathbf{S}') = \boldsymbol{\xi}' \otimes \boldsymbol{\xi}' \text{ with } \boldsymbol{\xi}' = \boldsymbol{\xi}_1 + \bar{\boldsymbol{\xi}}_1 = \boldsymbol{\xi}_1 + \sum_{p=2}^{n} c_p \cdot \boldsymbol{\xi}_p,$$

$$c_p = \frac{\boldsymbol{\xi}_p^T\boldsymbol{\eta}}{\alpha_p - \alpha_1}, \text{ and } \boldsymbol{\eta} = \bar{\mathbf{A}}\boldsymbol{\xi}_1.$$

□

Algorithm 6.2: IncSimFusion+ $(\mathcal{G}, \mathbf{A}, (\alpha_p, \boldsymbol{\xi}_p), \bar{\mathcal{G}}, (u, v))$

Input : Network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, the old UAM $\mathbf{A}$ of $\mathcal{G}$,
eigen-pairs $(\alpha_p, \boldsymbol{\xi}_p)$ of $\mathbf{A}$, the update $\bar{\mathcal{G}}$ to $\mathcal{G}$, query (u, v).

Output: New similarity score $s'(u, v)$.

compute the incremental UAM $\bar{\mathbf{A}}$ for the update $\bar{\mathcal{G}}$:
$\bar{\mathbf{A}} \leftarrow$ UpdateA $(\mathcal{G}, \mathbf{A}, \bar{\mathcal{G}})$;
initialize $a \leftarrow [\boldsymbol{\xi}_1]_u, \quad b \leftarrow [\boldsymbol{\xi}_1]_v$;
compute $\boldsymbol{\eta} \leftarrow \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1$;
free $\bar{\mathbf{A}}$, $\boldsymbol{\xi}_1$;
for $p \leftarrow 2, \cdots, n$ **do**
    compute $t \leftarrow \boldsymbol{\xi}_p^T \cdot \boldsymbol{\eta}, \quad c_p \leftarrow t/(\alpha_p - \alpha_1)$;
    compute $a \leftarrow a + c_p \times [\boldsymbol{\xi}_p]_u, \quad b \leftarrow b + c_p \times [\boldsymbol{\xi}_p]_v$;
    free α_p, $\boldsymbol{\xi}_p$;
free $\boldsymbol{\eta}$, t ;
compute $s'(u, v) \leftarrow a \times b$;
return $s'(u, v)$;

The main idea in incrementally computing $\mathbf{S}'$ is to reuse $\bar{\mathbf{A}}$ and the eigen-pair $(\alpha_p, \boldsymbol{\xi}_p)$ of the original $\mathbf{A}$. From the computational perspective, memorization techniques can be applied to Eq.(6.11) for an extra speed-up in computing $[\boldsymbol{\xi}']_i$. Once $\boldsymbol{\eta}$ is computed, it can be memorized for computing $c_2, \cdots, c_n$. When $c_2, \cdots, c_n$ are calculated, they can be memorized for computing $[\boldsymbol{\xi}']_i$ and $[\boldsymbol{\xi}']_j$.

6.5.2 An Incremental Algorithm for SimFusion+

We next prove Theorem 6.13 by providing an incremental algorithm, referred to as Inc-SimFusion+, for handling δ edge updates.

Algorithm. The algorithm accepts as input a network $\mathcal{G}$, the UAM $\mathbf{A}$ of $\mathcal{G}$, the eigen-pairs $(\alpha_p, \boldsymbol{\xi}_p)$ of $\mathbf{A}$, an update $\bar{\mathcal{G}}$ (a list of edge insertions and deletions) to $\mathcal{G}$, and a vertex

pair (u, v).

It works as follows. (a) IncSimFusion+ first computes the incremental UAM $\bar{\mathbf{A}}$ for the update $\bar{\mathcal{G}}$ by using procedure UpdateA (line 1). UpdateA incrementally finds all the changes to the old UAM $\mathbf{A}$ in the presence of a list of edge updates to $\mathcal{G}$. (b) For the given vertex pair (u, v), IncSimFusion+ initializes a and b based on the dominant eigenvector $\boldsymbol{\xi}_1$ of $\mathbf{A}$ (line 2) ; it computes $\boldsymbol{\eta}$ once and memorizes $\boldsymbol{\eta}$ for computing $c_2, \cdots, c_n$ (line 3). (c) Once computed, $c_2, \cdots, c_n$ are memorized for calculating the u-th and v-th entries of the dominant eigenvector $\boldsymbol{\xi}'$ of the new UAM,which is collected in a and b, respectively (lines 4-9). IncSimFusion+ returns $a \times b$ as the similarity $\hat{s}(u, v)$ (lines 10-11).

Edge Update. The procedure UpdateA is used for incrementally updating the UAM $\mathbf{A}$ by virtue of $\bar{\mathcal{G}}$. An update $\bar{\mathcal{G}}$ is represented as a sequence of 2-tuples $(\mathcal{D} \times \mathcal{D}, \mathsf{op})$ that records every single action of the edge update, in which $\mathcal{D} \times \mathcal{D}$ is a set of δ edges to be inserted or deleted, and op is either "+" (edge insertion) or "−" (edge deletion). For instance, after the edge (P_3, P_5) is added and (P_1, P_2) is removed from $\mathcal{G}_1$ in Fig.6.1, the update $\bar{\mathcal{G}}$ is denoted by

$$\bar{\mathcal{G}} = \{(P_3, P_5, +), (P_1, P_2, -)\}.$$

UpdateA identifies the incremental $\bar{\mathbf{A}}$ in two phases. (i) It first finds the affected nodes and the data spaces for each edge update in $\bar{\mathcal{G}}$ using a breadth-first search. (ii) It then updates the corresponding entries of $\bar{\mathbf{A}}$ based on the following. We abuse the notation $\mathcal{N}_{\mathcal{D}}(u)$ to denote all the neighbors of object u in the data space $\mathcal{D}$, *i.e.,*

$$\mathcal{N}_{\mathcal{D}}(u) = \{v \in \mathcal{D} |\ (u, v) \in \mathcal{R}\}.$$

Based on the partition of the entire data space $\mathcal{D} = \bigcup_{i=1}^{N} \mathcal{D}_i$ with $n_i = |\mathcal{D}_i|$, the incremental UAM $\bar{\mathbf{A}}$ can be accordingly partitioned into N^2 submatrices $\bar{\mathbf{A}}_{i,j}$.

- For each edge insertion $(u, v, +) \in \bar{\mathcal{G}}$ with $u \in \mathcal{D}_i$ and $v \in \mathcal{D}_j$, (i) we set all entries of $[\bar{\mathbf{A}}_{i,j}]_{u,\star}$ to $-\frac{\lambda_{i,j}}{n_j}$ except for their v-th entries to 0 if $\mathcal{N}_{\mathcal{D}_j}(u) = \varnothing$; (ii) we set all entries of $[\bar{\mathbf{A}}_{j,i}]_{\star,v}$ to $-\frac{\lambda_{j,i}}{n_i}$ except for their u-th entries to 0 if $\mathcal{N}_{\mathcal{D}_i}(v) = \varnothing$; (iii) we set $[\bar{\mathbf{A}}_{i,j}]_{u,v} = \lambda_{i,j}$ otherwise.

- For each edge deletion $(u, v, -) \in \bar{\mathcal{G}}$ with $u \in \mathcal{D}_i$ and $v \in \mathcal{D}_j$, (i) we set all entries of $[\bar{\mathbf{A}}_{i,j}]_{u,\star}$ to $\frac{\lambda_{i,j}}{n_j}$ except for their v-th entries to 0 if $|\mathcal{N}_{\mathcal{D}_j}(u)| = 1$; (ii) we set all entries of $[\bar{\mathbf{A}}_{j,i}]_{\star,v}$ to $\frac{\lambda_{j,i}}{n_i}$ except for their u-th entries to 0 if $|\mathcal{N}_{\mathcal{D}_i}(v)| = 1$; (iii) we set $[\bar{\mathbf{A}}_{i,j}]_{u,v} = -\lambda_{i,j}$ otherwise.

Complexity. The algorithm IncSimFusion+ is in $O(\delta n)$ time and $O(n)$ space for handling δ edge updates in $\bar{\mathcal{G}}$. (i) The procedure UpdateA can be bounded by $O(\delta n)$ time and $O(n)$ space (line 1). For each edge update in $\bar{\mathcal{G}}$, it is in at most $O(n)$ time and $O(n)$ intermediate space to update the corresponding entries of $\bar{\mathbf{A}}$. (ii) Computing $\boldsymbol{\eta}$ requires an $O(\delta)$-time and $O(n)$-space sparse matrix-vector multiplication $\bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1$ (line 3). (iii) For every c_p, it takes $O(\delta)$ time and $O(n)$ space to calculate $\boldsymbol{\xi}_p^T \cdot \boldsymbol{\eta}$ (line 6) since $\boldsymbol{\eta}$ is a sparse vector with only $O(\delta)$ nonzeros; and $c_2, \cdots, c_p$ are memorized for computing $[\boldsymbol{\xi}']_i$, which requires $O(\delta n)$ time and $O(n)$ space in total. (iv) Computing a, b and $s'(u, v)$ needs constant time and space (lines 7 and 10). Thus, combining (i)-(iv) , the total complexity is bounded by $O(\delta n)$ time and $O(n)$ space.

Example 6.15. We show how IncSimFusion+ works. Consider the graph $\mathcal{G}_1$ in Fig.6.1 with its UAM $\mathbf{A}$ in Example 6.3. Suppose two edges (P_1, P_2) and (P_2, P_1) are removed from $\mathcal{G}_1$. The new USM $\mathbf{S}'$ is updated as follows.

First, UpdateA is invoked for precomputing the incremental $\bar{\mathbf{A}}$ and the eigen-pairs $(\alpha_p, \boldsymbol{\xi}_p)$ of $\mathbf{A}$ in an off-line fashion:

$$\bar{\mathbf{A}} = \begin{bmatrix} 0 & -\frac{1}{6} & 0 & 0 & 0 \\ -\frac{1}{6} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

p	α_p	$\boldsymbol{\xi}_p$	c_p
1	1.184	$[.431\ .673\ .451\ .322\ .232]^T$	–
2	.503	$[.708\ -.522\ -.242\ .388\ .132]^T$	.062
3	-.480	$[-.256\ -.020\ .095\ .716\ -.641]^T$	-.018
4	-.366	$[-.021\ -.507\ .853\ -.119\ .017]^T$	-.025
5	.242	$[.497\ .127\ .037\ -.467\ -.719]^T$	.069

IncSimFusion+ next computes $\boldsymbol{\eta}$ from $\bar{\mathbf{A}}$ and $\boldsymbol{\xi}_1$ (line 3):

$$\boldsymbol{\eta} = \bar{\mathbf{A}} \cdot \boldsymbol{\xi}_1 = [-.112\ -.072\ 0\ 0\ 0]^T.$$

Then, c_p can be derived from the memorized $\boldsymbol{\eta}$ and $(\alpha_p, \boldsymbol{\xi}_p)$, *e.g.*,

$$c_2 = \boldsymbol{\xi}_2^T \cdot \boldsymbol{\eta}/(\alpha_2 - \alpha_1) = -.0419/(.503 - 1.184) = .062,$$
$$c_3 = \boldsymbol{\xi}_3^T \cdot \boldsymbol{\eta}/(\alpha_3 - \alpha_1) = .030/(-.480 - 1.184) = -.018.$$

Once computed, $c_2, \cdots, c_5$ are memorized for calculating $[\boldsymbol{\xi}']_\star$:

$$\boldsymbol{\xi}' = \boldsymbol{\xi}_1 + \sum_{p=2}^{5} c_p \times \boldsymbol{\xi}_p = [.327\ .703\ .485\ .326\ .266]^T.$$

Hence, applying $[\boldsymbol{\xi}']_\star$ to the new USM $\mathbf{S}'$ (line 10) yields

$$\mathbf{S}' = \begin{bmatrix} .107 & .230 & .159 & .107 & .087 \\ .230 & .494 & .341 & .230 & .187 \\ .159 & .341 & .235 & .158 & .129 \\ .107 & .230 & .158 & .107 & .087 \\ .087 & .187 & .129 & .087 & .071 \end{bmatrix}. \quad \square$$

6.6 Experimental Evaluation

In this section, a comprehensive empirical study of the proposed similarity estimating methods is presented.

6.6.1 Experimental Setting

Datasets. We used three real-life datasets and a synthetic dataset.

(1) MSN *Data.* [7] The MSN search log data were taken from "Microsoft Live Labs: Accelerating Search in Academic Research". This dataset was also used in the prior work [XFF+05]. It contains about 15M user queries from the United States in May 2006 and the corresponding clickthrough URLs. The dataset was formatted by showing each query, the URLs of the associated web pages, and the number of clickthroughs by query, as depicted below.

Query	URL	Clicks	URL	Clicks
Shopping	shopping.yahoo.com	2,375	www.ebay.com	1,859

[7] http://research.microsoft.com/ur/us/fundingopps/RFPs/Search_2006_RFP.aspx

The 15K most common queries in the search log were chosen, and the hyperlinks from the contents of the top 32K popular web pages were parsed. We built a network $\mathcal{G} = (\mathcal{D}, \mathcal{R})$, which consists of a web page space $\mathcal{D}_w$ and a query space $\mathcal{D}_q$.

(2) DBLP *Data.* [8] This dataset was derived from a snapshot of the computer science bibliography (from 2001 to 2010). We selected the research papers published in the following conference proceedings: "SIGIR", "KDD", "VLDB", "ICDE", "SIGMOD" and "WWW". Choosing a time step of two years, we built 5 DBLP web graphs $\mathcal{G}_i$ $(i = 1, \cdots, 5)$ with the sizes listed below:

	$\mathcal{G}_1$: 01-02	$\mathcal{G}_2$: 01-04	$\mathcal{G}_3$: 01-06	$\mathcal{G}_4$: 01-08	$\mathcal{G}_5$: 01-10
$\lvert\mathcal{D}\rvert$	1,838	3,723	5,772	9,567	12,276
$\lvert\mathcal{R}\rvert$	7,103	14,419	29,054	45,310	64,208

For each graph $\mathcal{G}_i = (\mathcal{D}_i, \mathcal{R}_i)$, two data spaces were used: paper space $\mathcal{D}_p^i$ and author space $\mathcal{D}_a^i$.

(3) WebKB *Data.* [9] This dataset collects web pages from the computer science departments of four universities: Cornell (CO), Texas (TE), Washington (WA) and Wisconsin (WI). It was also used in the previous work [CZDC10] for link-based similarity estimation. For each university, a network $\mathcal{G}_{U_i} = (\mathcal{D}_i, \mathcal{R}_i)$ was built, in which (a) the web pages in $\mathcal{D}_i$ were classified into 7 categories (data spaces): student, faculty, staff, department, course, project and others, and (b) the UAM of $\mathcal{R}_i$ represented the hyperlink adjacency matrix. The sizes of these networks are as follows:

	U_1: CO	U_2: TE	U_3: WA	U_4: WI
$\lvert\mathcal{D}\rvert$	867	827	1,263	1,205
$\lvert\mathcal{R}\rvert$	1,496	1,428	2,969	1,805

(4) Synthetic Data. The data were produced by the C++ boost graph generator, with 2 parameters: the number of vertices and the number of edges. Varying the graph parameters, we used this dataset to represent homogenous networks for an in-depth analysis.

[8] http://www.informatik.uni-trier.de/~ley/db/
[9] http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/

Compared Algorithms. The following algorithms were implemented in C++: (1) SimFusion+ and IncSimFusion+ ; (2) SF, a SimFusion algorithm via matrix iteration [XFF+05]; (3) CSF, a variant of SF, which leverages PageRank stationary distribution [CZDC10]; (4) SR, a SimRank algorithm via partial sums function [LVGT10]; (5) PR, a Penetrating-Rank algorithm encoding both in- and out-links [ZHS09].

Evaluation Metrics. For evaluating the performance of the algorithms, we used *Normalized Discounted Cumulative Gain* (NDCG) metrics [CZDC10]. The NDCG at a rank position p is defined as

$$\text{NDCG}_p = \frac{1}{\text{IDCG}_p} \sum_{i=1}^{p} \frac{2^{\text{rank}_i} - 1}{\log_2 (1 + i)},$$

where rank_i is the graded relevance of the similarity result at rank position i, and IDCG_p is the normalization factor to guarantee that NDCG of a perfect ranking at position p equals 1.

Twelve IT experts were hired to judge the similarity of the five algorithms. The final judgment was rendered by a majority vote.

All experiments were run on a machine with a Pentium(R) Dual-Core (2.00GHz) CPU and 4GB RAM, using Windows Vista. The algorithms were implemented in Visual C++. Each experiment was repeated over 5 times, and the average is reported here.

6.6.2 Experimental Results

Exp-1: Accuracy

We first evaluated the accuracy of SimFusion+ *vs.* SF, CSF, SR and PR in estimating the similarity, using real-life data.

We randomly chose 50 queries and 40 pages from the MSN query log, and compared the average NDCG_{10} (and NDCG_{30}) of the five algorithms. The results are shown in Figure 6.4, in which the x-axis categorizes the objects according to query and web page. We find the following. (i) In most cases, SF seems hardly to get sensible similarities because with the increasing number of iterations, all the similarities of SF will asymptotically approach the same value. This verifies the convergence issue of the original

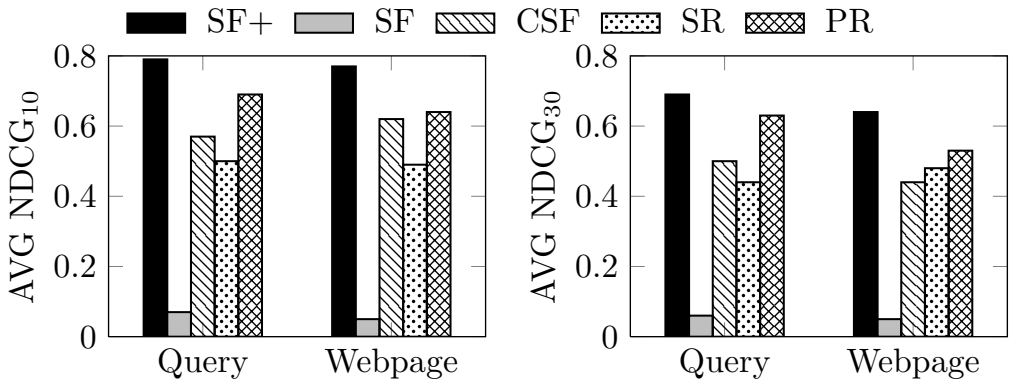


Figure 6.4: $NDCG_{10}$ and $NDCG_{30}$ on MSN

model [XFF+05]. (ii) When SF did not fail, SimFusion+ always gave more accurate estimation on average than the other algorithms. For instance, for the top 10 queries, the average $NDCG_{10}$ of SimFusion+ (0.79) is 10x better than SF (0.07), 39% better than CSF (0.57), 58% better than SR (0.50), and 15% better than PR (0.69), whereas for the top 30 web pages, the average $NDCG_{30}$ of SimFusion+ (0.64) is 12x better than SF (0.05), 45% better than CSF (0.44), 33% better than SR (0.48), and 21% better than PR (0.53). This is because substituting UAM for URM effectively avoids divergent or trivial solutions, thus improving the quality and reliability of SimFusion+ similarity.

To further verify the accuracy, we randomly selected another 15 queries and 15 web pages from MSN data. In Figure 6.5, the query-by-query and page-by-page comparisons are shown for $NDCG_{10}$ of the five algorithms. We observe that (i) for 12 out of 15 queries, SimFusion+ achieved highest accuracy of the five algorithms;for 13 out of 15 web pages, SimFusion+ outperformed the other algorithms in its accuracy, and was slightly less accurate than PR for only 2 pages. (ii) For all the queries and web pages, SimFusion+ showed the best accuracy performance on average, PR the second, and SF the worst. This is because SimFusion+ uses UAM to encode the intra- and inter-relations in a comprehensive way, thus making the results unbiased.

We also evaluated the performance of SimFusion+ on DBLP and WebKB datasets. In DBLP experiments, 20 authors were randomly chosen from $\mathcal{G}_1$:01-02, $\mathcal{G}_3$:01-06 and $\mathcal{G}_5$:01-10 data, respectively. We compared the similarity of the top 10 authors in $\mathcal{G}_i$ ($i =$

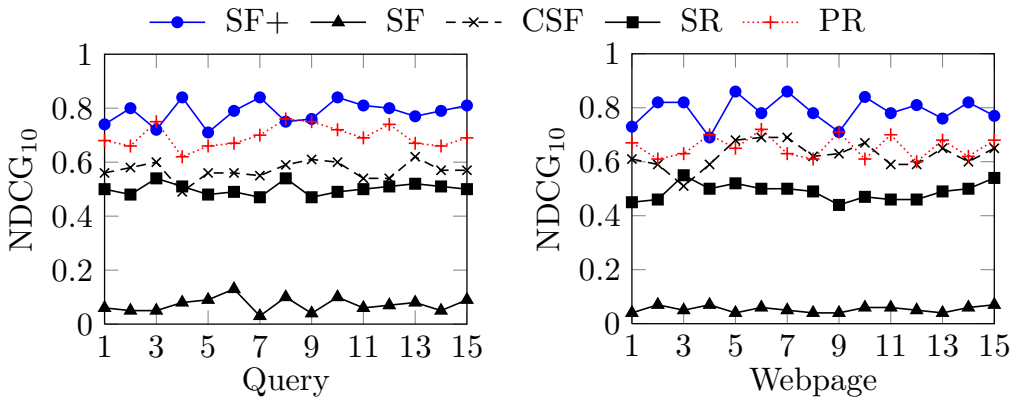


Figure 6.5: Query-by-query and page-by-page comparisons for $NDCG_{10}$ on MSN

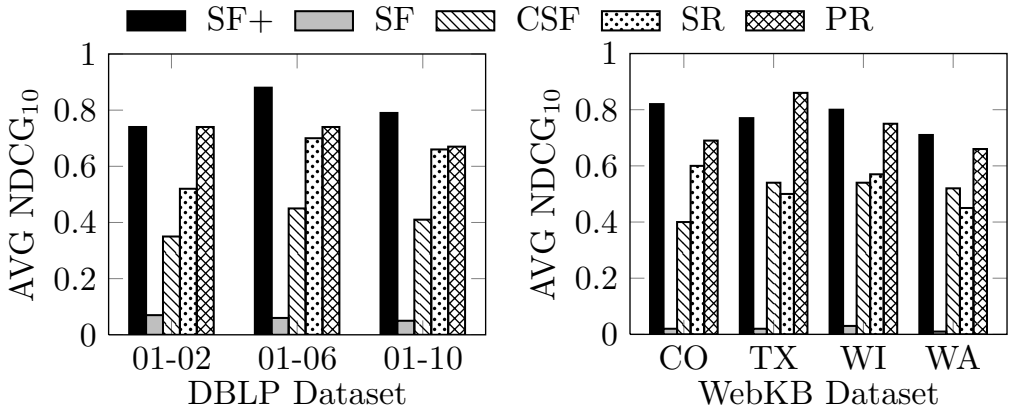


Figure 6.6: $NDCG_{10}$ on DBLP and WebKB

$1, 3, 5$) estimated by the five algorithms. The results of the average $NDCG_{10}$ are depicted in Figure 6.6. It can be seen that SimFusion+ again achieved better accuracy on DBLP data. For instance, SimFusion+ (0.88) on $\mathcal{G}_3$:01-06 was 13x better than SF (0.06), 95% better than CSF (0.45), 26% better than SR (0.7), and 19% better than PR (0.74). In WebKB experiments, we computed NDCG within 10 web pages for each object in each university data (CO,TE,WI,WA) and evaluated the average scores. Figure 6.6 shows that SimFusion+ outperformed the other 4 algorithms on CO, WI and WA data, except that PR (0.8) did 6% better than SimFusion+ (0.75) on TX data. This tells that SimFusion+ accuracy performance is consistently stable on different experimental datasets.

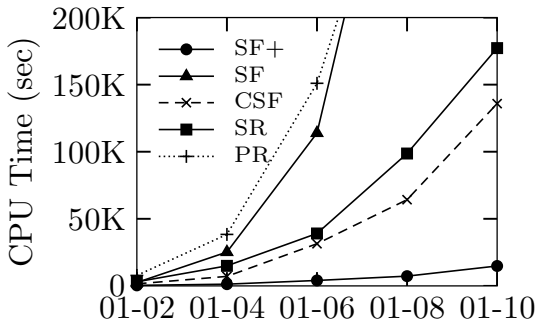


Figure 6.7: Running Time on DBLP

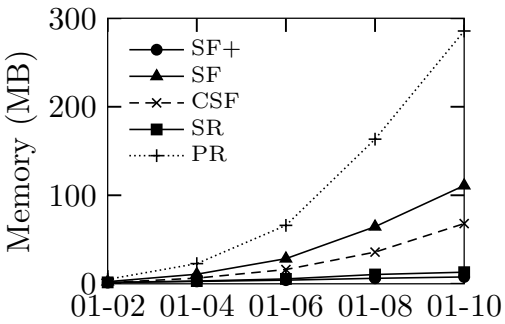


Figure 6.8: Memory Space on DBLP

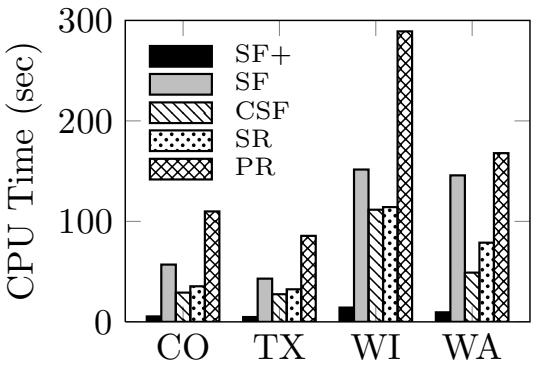


Figure 6.9: Running Time on WebKB

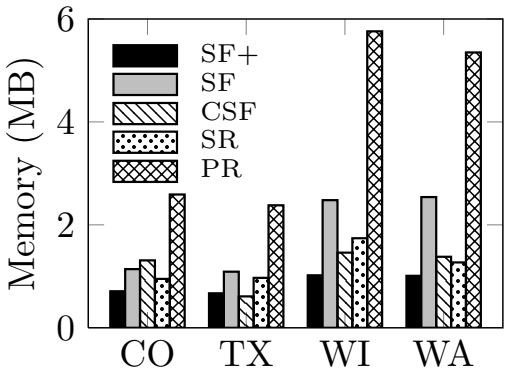


Figure 6.10: Memory Space on WebKB

Exp-2: CPU Time & Memory Space

We then evaluated the running time and memory space efficiency of SimFusion+, SF, CSF, SR and PR using real datasets.

Figures 6.7 and 6.8 show the CPU time and memory consumption for the five algorithms on DBLP. The total time and memory for each algorithm showed an increasing tendency with the growing size of DBLP. It can be noted that the time for SimFusion+ was at least one order of magnitude faster than CSF and SR on average, and more than 20x faster than PR and SF, whereas the space for SimFusion+ and SR increased linearly with the size of DBLP, in contrast with the quadratic increase in memory for SF, CSF, PR, as expected. This drastic speedup and decrease in RAM is due to the memorization of $\sigma_{\max}(\mathbf{T}_k)$ for computing USM, thus saving much time and space for repetitive matrix multiplications.

To further evaluate the efficiency, we compare the time and memory of the five ranking

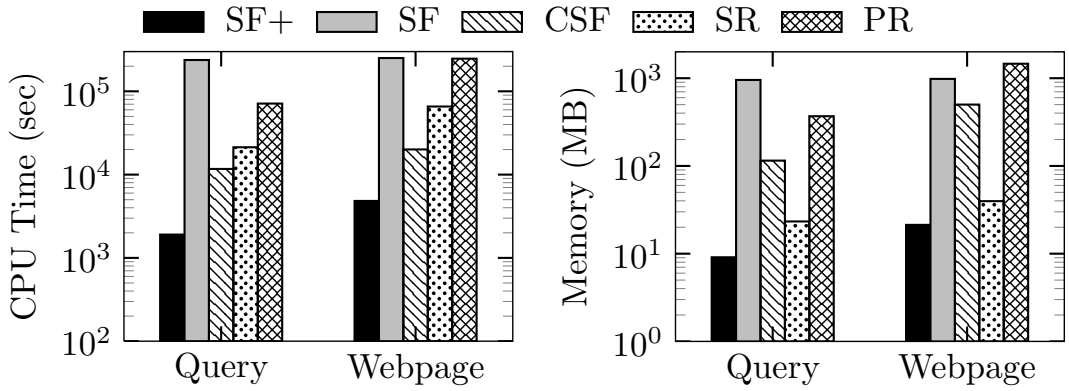


Figure 6.11: CPU time and memory for the given query and web page on MSN

algorithms on WebKB. In Figures 6.9 and 6.10, the results indicate that SimFusion+ took about 10x less time than SF and PR, and 6x less time than CSF and SR on average. For instance, the CPU time for SimFusion+ (14.2s) on WI data is 9.7x faster than SF (151.6s), 7x faster than CSF (111.6s) and SR (114.2s), and 19x faster than PR (289.1s). The memory space for SimFusion+ was also efficient and scaled well with the size of WebKB. It can be seen that SF also took small memory space (approx. 1M) when the data size was small (*e.g.,* CO and TX). However, when the data size increased (*e.g.,* WI and WA), SF was less useful since large memory storage (about 2.5M) was required to keep the intermediate result of the k-th iterative USM. In all the cases, SimFusion+ performed the best.

On large datasets, the effect of SimFusion+ is even more pronounced. Figure 6.11 reports the average time and memory of the five algorithms on MSN data, in which the y-axis is log-scale. We chose 50 queries and 40 web pages from $\mathcal{D}_q$ and $\mathcal{D}_w$, respectively. For each $o_q \in \mathcal{D}_q$ ($resp.o_w \in \mathcal{D}_w$), we estimated the similarity between o_q ($resp.o_w$) and other object $o \in \mathcal{D}_q \cup \mathcal{D}_w$. We see that SimFusion+ was highly efficient on large datasets (nearly 2 orders of magnitude than SF and PR, and 1 order of magnitude than CSF in both time and space). This validates that the performance of SimFusion+ is fairly stable among different datasets.

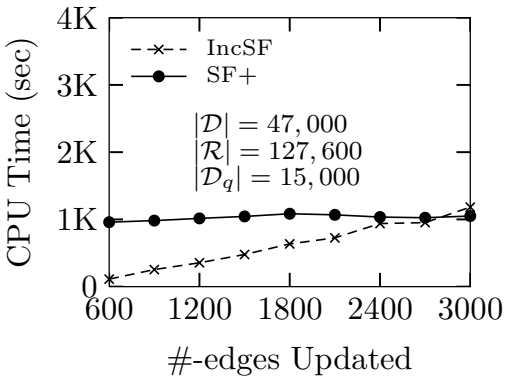


Figure 6.12: IncSimFusion+ for query

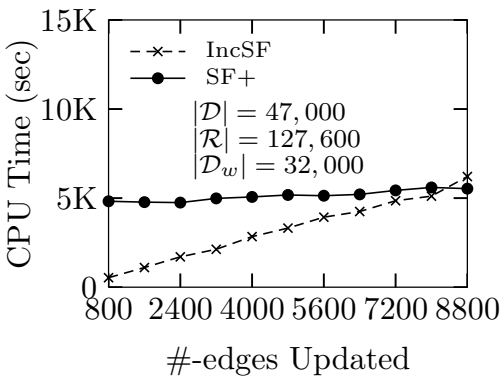


Figure 6.13: IncSimFusion+ for web page

Exp-3: Incremental Performance

We next evaluated the incremental performance of IncSimFusion+ on real datasets. Given a sequence of edge updates (δ edge insertions and deletions in $\bar{\mathcal{G}}$), we compared the running time of IncSimFusion+ with that of SimFusion+ ; the latter had to recalculate the UAM when edges were updated, and the computational cost was counted. In these experiments, the eigen information of the old UAM can be preconditioned in an off-line fashion and shared by all the updated graphs for incremental computation, and hence their costs were not counted at the query stage. Due to space limitations, below we only reported the results on MSN dataset.

Varying δ (the number of the edges to be updated) from 600 to 3000, we first evaluated the running time of IncSimFusion+ and SimFusion+ over MSN data, respectively, for estimating all the similarities between the query objects in $\mathcal{D}_q$. Figure 6.12 shows that IncSimFusion+ outperformed SimFusion+ when $\delta < 2800$, but SimFusion+ performed better for larger δ, as expected. This is because the small value of δ often preserves the sparseness of the incremental UAM, and hence reduces the computational cost of $\boldsymbol{\eta}$ when the USM was incrementally updated.

To further validate the performance of IncSimFusion+ , we estimated the similarity among the web page in $\mathcal{D}_w$ and tested its CPU time on MSN data. In Figure 6.13, the result shows that IncSimFusion+ was highly efficient for the small number of edge updates. When $\delta > 7700$, SimFusion+ did better than IncSimFusion+ . This tells that

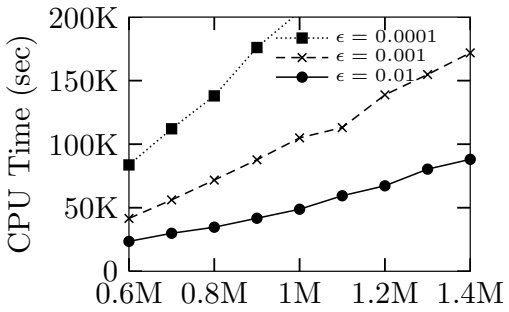


Figure 6.14: Effect of ϵ on Time

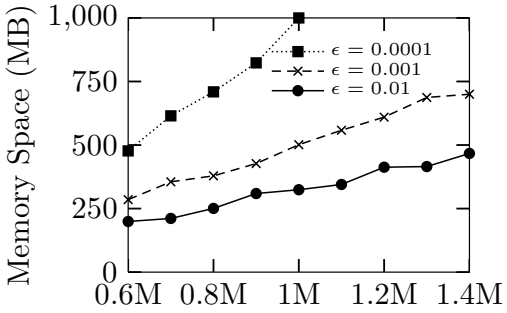


Figure 6.15: Effect of ϵ on Memory

increasing the number of updated edges induces more nonzeros in $\bar{\mathbf{A}}$, thus increasing the difficulty of incremental computation. We also noticed that the SimFusion+ time was less sensitive to the small number of updated edges, whereas the IncSimFusion+ time was linearly increased with δ, as expected. This is because once the edges are changed, SimFusion+ has to recompute all the similarities from scratch. In contrast, IncSimFusion+ only computes the similarities from the affected area of edge updates. In light of this, IncSimFusion+ scales well with δ.

Exp-4: Effect of ϵ

We used 9 web graphs with the number of vertices increased from 600K to 1.4M. We varied ϵ from 0.01 to 0.0001 and ran SimFusion+ on each graph. The results are reported in Figures 6.14 and 6.15. which visualizes the time and space of SimFusion+ needed for estimating similarities. The x-axis gives the number of vertices for each graph. It can be seen that the computational time and memory consumption for SimFusion+ was sensitive to ϵ. The smaller the ϵ is, the larger amounts of the CPU time and memory space are, as expected. These confirmed our observation in Section 6.4, where we envisage that the small choice of ϵ imposes more iterations on computing $\mathbf{T}_k$ and $\mathbf{v}_k$, and hence increases the estimation costs.

6.7 Related Work

The link-based similarity has become increasingly popular since the famous result of Google PageRank [PBMW99] on ranking web pages. Since then, there has been a surge of papers focusing on web link analysis. In particular, a growing interest has been witnessed in the SimFusion model over the past decade [XZC+04a, XFF+05] as it provides a useful measure of similarity that supports different kinds of intra- and inter-node relations from multiple data spaces.

The iterative computation of SimFusion was proposed in [XFF+05] with several problems left open there. In comparison, this work extends [XFF+05] by (i) addressing the divergent and trivial solution of the original SimFusion, (ii) optimizing the time and space complexity of similarity computation, and (iii) supporting incremental update on evolving graphs, none of which was considered in [XFF+05].

It is worth mentioning that Jeh and Widom have proposed a similar structural measure called SimRank [JW02], predicated, as SimFusion is, on the idea that vertices are similar if they have similar neighbor structures. The essential difference between the two models is the notion of the convergence principle. SimFusion ensures the existence of the stationary distribution and ergodicity convergence to this distribution, whereas SimRank hinges on a damping factor $0 < c < 1$ to govern the convergence.

Optimization techniques have been devised for SimRank computation (*e.g.,* [LVGT10, LHH+10, HLC+12, YZL+12]). The best-known SimRank algorithm yields $O(k \min\{nm, \frac{n^3}{\log n}\})$ time [LVGT10]. The performance gain is mainly achieved by a partial sum function for amortization; as for SimFusion, the conventional matrix multiplication in its iterative formula misled its complexity, which was previously considered $O(kn^3)$ time and $O(n^2)$ space. The idea of the dominant eigenvector in this work significantly improves its computation to $O(km)$ time and $O(kn)$ space, which is more efficient than SimRank [LVGT10].

There has also been work on link-based similarity computation. A unified framework of link-based analysis was addressed in [XZC+04a], which extends PageRank and HITS by (i) considering both inter- and intra-type relationships, and (ii) bringing order to

data objects in different data spaces. It differs from this work in that the focus is on finding attribute values of a single object, rather than on improving the complexities for similarity estimation. Extensions of similarity reinforcement assumption were studied in [XZC+04b], by spreading multiple relationship similarities over interrelated data objects to enhance their mutual reinforcement effects. Nevertheless, neither of these deduces rigorous mathematical formulae, and the rationales behind the integration approaches are different from this work. Recently, a closed-form solution to P-Rank (Penetrating-Rank) formula was addressed in [CZDC10]. Cai *et al.* [CZDC10] showed that when the damping factor $c = 1$ and weighting factor $\gamma = 0$, P-Rank can be reduced to SimFusion. However, this reasoning is based on the flawed assumption that the diagonal entries diag($\mathbf{S}$) of the P-Rank similarity matrix were not considered. We argue that P-Rank is defined recursively, and hence, the omission of diag($\mathbf{S}$) has an impact on the similarity of a vertex with itself, and recursively, it has an impact on the similarity of different pairs of vertices.

6.8 Conclusions

In this chapter, we have presented SimFusion+, a revision of SimFusion, for preventing the trivial solution and the divergence issue of the SimFusion model. We proposed efficient techniques to improve the time and space complexity of SimFusion+ computation with accuracy guarantees. We also devised an incremental algorithm to compute SimFusion+ similarity on dynamic graphs when edges are frequently updated. The empirical results on both real and synthetic datasets showed that our methods can achieve high performance and result quality. We are currently studying the vertex-updating methods for incrementally computing SimFusion+. We are also to extend our techniques to parallel SimFusion+ computing on GPU.

Chapter 7

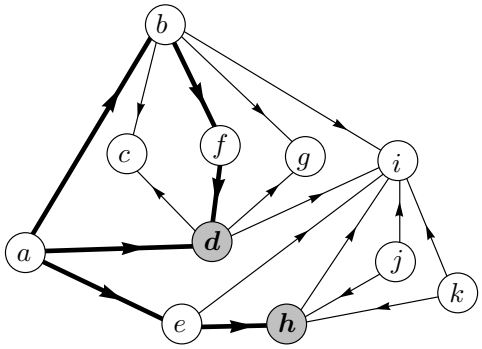
A Novel Model for Node-Pair Relevance Assessment

7.1 Introduction

The task of assessing similarity between two nodes based on hyperlinks is a long-standing problem in information search. However, it is a complex challenge to find an appropriate link-based scoring function since a satisfactory general-purpose similarity measure should better simulate human judgement behavior, with simple and elegant formulations [LVGT08].

7.1.1 Motivation

Recently, SimRank [JW02] has received growing interest as a widely-accepted measure of similarity between two nodes. While significant efforts have been devoted to optimizing SimRank computation (*e.g.,* [FR05, HLC+12, LHH+10, LVGT08]), the semantic issues of SimRank have attracted little attention. We observe that SimRank has an undesirable property, namely, "zero-similarity": SimRank score $s(i, j)$ only accommodates the paths with *equal length* from a common "source" node to both i and j. Thus, other paths for node-pair (i, j) are fully ignored by SimRank. as shown in Example 7.1.



Node-Pairs	SR	PR	SR*	RWR
(h, d)	0	.049	.010	0
(a, f)	0	.075	.032	.032
(a, c)	0	0	.025	.024
(g, a)	0	0	.025	0
(g, b)	0	0	.075	0
(i, a)	0	0	.015	0
(i, h)	.044	.041	.031	0

Figure 7.1: Similarities on Citation Graph

Example 7.1. Consider a citation network $\mathcal{G}$ in Figure 7.1, where each node represents a paper, and an edge a citation. Using the damping factor $C = 0.8$ [1], we compute SimRank similarity of node-pairs in $\mathcal{G}$. It can be noticed that many node-pairs in $\mathcal{G}$ have zero SimRank when they have no in-coming paths of equal length from a common "source" node, as partly depicted in Column 'SR' of the table. For instance, $s(h, d) = 0$ as the in-link "source" a is not in the center of the paths: $h \leftarrow e \leftarrow \boxed{a} \rightarrow d$ [2], $h \leftarrow e \leftarrow \boxed{a} \rightarrow b \rightarrow f \rightarrow d$, meaning that when we recursively compute the similarity of the in-neighbors prior to computing the similarity of the two nodes themselves, there is no likelihood for this recursion to reach the base case (a common in-link "source") that a node is maximally similar to itself. Similarly, $s(a, g) = 0$ as a has no in-neighbors, not to mention the fact that there is no such in-link "source" with equal distance to both a and g. In contrast, $s(g, i) > 0$ as there is an in-link "source" b (*resp.*d) in the center of $g \leftarrow \boxed{\text{b}} \rightarrow i$ (*resp.*$g \leftarrow \boxed{\text{d}} \rightarrow i$). □

The "zero-SimRank" phenomenon in Example 7.1 is rather counter-intuitive. An evident example is $s(h, d) = 0$. We note in Figure 7.1 that h and d do have a common in-link "source" a, just except for the *equal-length* distance from a to both h and d. Hence, h and d should have some relevance. Another example is a path graph of length $2n$ as follows: $a_{-n} \leftarrow \cdots \leftarrow a_{-1} \leftarrow \boxed{a_0} \rightarrow a_1 \rightarrow \cdots \rightarrow a_n$, where each a_i $(i =$

[1] As suggested in [JW02], C is empirically set around 0.6–0.8, which gives the rate of decay as similarity flows across edges.

[2] We abuse the notation $h \leftarrow e \leftarrow \boxed{a} \rightarrow d$ to denote the path of length 3, starting from h, taking 2 steps against the edge direction and 1 step along it, and finally arriving at d.

$0, \pm 1, \cdots, \pm n$) denotes a node. We notice that the SimRank $s(a_i, a_j) = 0$, for all $|i| \neq |j|$, which is quite against intuition since a_0 is the common root of all nodes a_i ($i = \pm 1, \cdots, \pm n$). As will be shown in Section 7.2, SimRank does neglect all contributions of in-link paths without a "source" node in the center, and the "zero-similarity" issue refers not only to the problem that SimRank may produce "completely zero scores" *(i.e., "completely dissimilar" issue)*, but also to the problem that SimRank may miss the contributions of a large class of in-link paths (even though their scores are not zero) due to the "zero contributions" of such paths to SimRank scores *(i.e., "partially missing" issue)*. Indeed, as demonstrated by our experiments in Figure 7.11, both scenarios of "zero-similarity" *commonly* exist in real graphs, *e.g.,* on CitHepTh, 95+% node-pairs have "zero-SimRank" issues, among which 40+% are assessed as "completely dissimilar", and 55+% (though SimRank $\neq$0) "partially miss" contributions of many paths, adversely affecting assessment quality. These motivate us to revise the existing SimRank model.

A pioneering piece of work by Zhao *et al.* [ZHS09] proposes rudiments of a novel approach to refining the SimRank model. Observing that SimRank may incur some unwanted "zero-similarities", they suggested P-Rank, an extension of SimRank, by taking both in- and out-links into consideration for similarity assessment, as opposed to SimRank that merely considers in-links. Although P-Rank, to some degree, might reduce "zero-similarity" occurrences in practice, we argue that such a "zero-similarity" issue arises, not because of a biased overlook of SimRank against out-links, but because of the blemish in SimRank philosophy that may miss the contribution of a certain kind of paths (whose in-link "source" is not in the center). In other words, P-Rank can not, in essence, resolve the "zero-similarity" issue of SimRank. For instance, nodes h and d are similar in the context of P-Rank, as depicted in Col. 'PR' of Figure 7.1, since there is an out-link "source" i in the center of the outgoing path $h \rightarrow \boxed{i} \leftarrow d$. However, if the edge $h \rightarrow i$ is replaced by $h \rightarrow l \rightarrow i$ with l being an inserted node, then the P-Rank of (h, d) is still zero, since in this case neither in- nor out-link "source" exists in the center of any incoming or outgoing paths of (h, d).

Our goal in this work is to propose an alternative model that can remedy SimRank

"zero-similarity" issues in nature, while inheriting merits of the basic SimRank philosophy. Keeping with an elegant form and to support fast clustering strategies, our model is intended to be a refinement of SimRank for semantic richness, and takes into account contributions of many incoming paths (whose common "source" is not strictly in the center) that are neglected by SimRank. The major challenge with establishing this model is that it is notoriously difficult to effectively assess $s(a, b)$ by finding out *all* the possible incoming paths between a and b, regardless of whether there exists a common "source" with equal distance to both a and b. This problem is hard because such a task often requires traversing far more possible incoming paths to fetch the similarity information, which might not only destroy the simplicity of the original SimRank formulation, but also increase the computational difficulty of the model. Fortunately, we observe that our model can be "purified" as a fairly elegant closed form, and there are opportunities for the new model to assess similarities without suffering from high computational costs.

7.1.2 Chapter Outlines

In the chapter, our main contributions are as follows.

- We propose SimRank*, a revision of SimRank, and justify its semantic richness. Our model provides a natural way of traversing more incoming paths that are largely ignored by SimRank for each node-pair, and thus enables counter-intuitive "zero-SimRank" nodes to be similar while inheriting the beauty of the SimRank philosophy. (Section 7.2)

- We show that the series form of SimRank* can be simplified into an elegant closed form, which looks more succinct yet has richer semantics than SimRank, without suffering from increased computational cost. This provides an iterative paradigm for computing SimRank* in $O(Knm)$ time on a graph of n nodes and m edges for K iterations, which is comparable to SimRank. (Subsects. 7.3.1–7.3.2)

- To further speed up SimRank* computation, as the existing technique [LVGT08] of partial sums memoization for SimRank optimization no longer applies, we leverage

a novel clustering approach for SimRank* via edge concentration. Due to its NP-hardness, an efficient algorithm is devised to improve SimRank* computation to $O(Kn\tilde{m})$ time, where $\tilde{m}$ is generally much smaller than m. (Subsect. 7.3.3)

We evaluate the performance of SimRank* on real and synthetic data. The results show that (i) SimRank* achieves higher quality of similarity assessment, as compared with the state-of-the-art SimRank [LVGT08], P-Rank [ZHS09] and RWR [TFP06]; (ii) Regarding computational efficiency, our algorithms are consistently faster than the baselines by several times.

7.2 SimRank*: A Revision of SimRank

We first show that the "zero-similarity" issue (Example 7.1) is *rooted* in both SimRank and non-SimRank based metrics. We then propose our treatment, SimRank*, for this semantic problem.

7.2.1 "Zero-SimRank" Issue

We shall abuse the following notions. (i) An *in-link path* ρ of node-pair (a, b) in $\mathcal{G}$ is a walk of length $(l_1 + l_2)$, denoted as $a = v_0 \leftarrow v_1 \leftarrow \cdots \leftarrow \boxed{v_{l_1}} \rightarrow v_{l_1+1} \rightarrow \cdots \rightarrow v_{l_1+l_2} = b$,[3] starting from a, taking l_1 steps against the directions of the edges $v_{i-1} \leftarrow v_i$ for every $i \in [1, l_1]$, and l_2 steps along the directions of $v_{i-1} \rightarrow v_i$ for every $i \in [l_1 + 1, l_1 + l_2]$, and finally arriving at b. (ii) The node v_{l_1} is called the *in-link "source"* of ρ. (iii) The *length* of in-link path ρ, denoted by $\mathsf{len}(\rho)$, is $(l_1 + l_2)$, *i.e.,* the number of edges in ρ.

Definition 7.2. An in-link path ρ is *symmetric* if $l_1 = l_2$. □

Definition 7.3. An in-link path ρ is *unidirectional* if $l_1 = 0$ or $l_2 = 0$. □

For example in Figure 7.1, $\rho : h \leftarrow e \leftarrow \boxed{a} \rightarrow d$ is an in-link path of node-pair (h, d), with a being its in-link "source". $\mathsf{len}(\rho) = 2 + 1 = 3$. ρ is not symmetric since $l_1 = 2 \neq 1 = l_2$.

[3] We allow a path from the "source" node to one end with repeated nodes to suit the existence of cycles in a graph.

Clearly, in-link path ρ is *symmetric* if and only if there is an in-link "source" in the center of ρ. Any in-link path of *odd* length (*i.e.*, $l_1 + l_2$ is odd) is dissymmetric.

"Zero-SimRank" Issue. Based on the notion of in-link paths, we next show the "zero-SimRank" issue as follows:

Theorem 7.4. *For any two distinct nodes a and b in $\mathcal{G}$, the SimRank score $s(a,b) = 0$ if there does not exist any symmetric in-link path of node-pair (a,b). More importantly, even if $s(a,b) \neq 0$, SimRank $s(a,b)$ may still "partially miss" all the contributions of dissymmetric in-link paths for (a,b).* □

As a proof of the theorem, we first extend the power property of an adjacency matrix. We then reinterpret SimRank based on its power series representation.

Extension of $\mathbf{A}^l$. Let $\mathbf{A}$ be the adjacency matrix of $\mathcal{G}$. There is an interesting property of $\mathbf{A}^l$ [BC08a]: The entry $[\mathbf{A}^l]_{i,j}$[4] counts the number of paths of length l from node i to j. Such a property can be readily generalized as follows:

Lemma 7.5. Let ρ be a "specific path" of length l, consisting of a sequence of nodes $i = v_0, v_1, \cdots, v_l = j$ with each edge being directed (1) from v_{k-1} to v_k, or (2) from v_k to v_{k-1}. Let $\bar{\mathbf{A}} = \prod_{k=1}^{l} \mathbf{A}_k$ with (1) $\mathbf{A}_k = \mathbf{A}$ if $\exists\ v_{k-1} \rightarrow v_k$ in ρ, or (2) $\mathbf{A}_k = \mathbf{A}^T$ if $\exists\ v_{k-1} \leftarrow v_k$ in ρ, for each $k \in [1,l]$. Then, the entry $[\bar{\mathbf{A}}]_{i,j}$ counts the number of specific paths ρ in $\mathcal{G}$. □

Lemma 7.5 can be proved by induction on l, which is similar to the proof of the power property of the adjacency matrix [BC08a, pp.51]. We omit it here due to space limits.

Lemma 7.5 allows counting the number of "specific paths" whose edges are not all necessarily in the same direction. For instance, for the path $\rho : i \rightarrow \circ \leftarrow \circ \rightarrow \circ \rightarrow \circ \leftarrow j$ with $\circ$ denoting any node in $\mathcal{G}$, we can build $\bar{\mathbf{A}} = \mathbf{A}\mathbf{A}^T\mathbf{A}\mathbf{A}\mathbf{A}^T$, in which $\mathbf{A}$ (*resp.*$\mathbf{A}^T$) is at the positions 1,3,4 (*resp.*2,5), corresponding to the positions of $\rightarrow$ (*resp.*$\leftarrow$) in ρ. Then, $[\bar{\mathbf{A}}]_{i,j}$ tallies the number of paths ρ in $\mathcal{G}$. If no such paths, $[\bar{\mathbf{A}}]_{i,j} = 0$. As another example, $[(\mathbf{A}^T)^{l_1} \cdot \mathbf{A}^{l_2}]_{i,j}$ tallies the number of in-link paths of length $(l_1 + l_2)$ for node-pair (i,j).

[4] In the sequel, $[\mathbf{X}]_{i,j}$ denotes the (i,j)-entry of matrix $\mathbf{X}$.

When all $\mathbf{A}_k$ ($\forall k \in [1, l]$) are set to $\mathbf{A}$, Lemma 7.5 reduces to the conventional power property of an adjacency matrix.

One immediate consequence of Lemma 7.5 is as follows:

Corollary 7.6. $\sum_{k=1}^{\infty} [(\mathbf{A}^T)^k \cdot \mathbf{A}^k]_{i,j}$ counts the total number of all symmetric in-link paths of node-pair (i, j) in $\mathcal{G}$. □

Corollary 7.6 implies that if there are no nodes with equal distance to both i and j (*i.e.,* if no symmetric in-link paths for node-pair (i, j)), then $[(\mathbf{A}^T)^k \cdot \mathbf{A}^k]_{i,j} = 0$, $\forall k \in [1, \infty)$.

SimRank Reinterpretation. Leveraging Corollary 7.6, we show why SimRank has "zero-similarity" issue: $s(i, j) = 0$ if there are no nodes with equal distance to both i and j.

We first rewrite SimRank matrix $\mathbf{S}$ as a power series.

Lemma 7.7. The SimRank $\mathbf{S}$ in Eq.(2.2) can be rewritten as

$$\mathbf{S} = (1 - C) \cdot \sum_{l=0}^{\infty} C^l \cdot \mathbf{Q}^l \cdot (\mathbf{Q}^T)^l. \qquad \square \tag{7.1}$$

Proof. According to [LHH+10, Eq.(4)], $\mathbf{S}$ has the closed form:

$$vec(\mathbf{S}) = (1 - C) \cdot (\mathbf{I}_n - C(\mathbf{Q} \otimes \mathbf{Q}))^{-1} vec(\mathbf{I}_n),$$

where $vec(\star)$ is a vectorization operator, $\otimes$ a tensor product.

Since $\|\mathbf{Q} \otimes \mathbf{Q}\|_\infty \leq 1$, the identity $(\mathbf{I}_n - \mathbf{X})^{-1} = \sum_{k=0}^{\infty} \mathbf{X}^k$ implies that

$$vec(\mathbf{S}) = (1 - C) \cdot \sum_{k=0}^{\infty} C^k (\mathbf{Q} \otimes \mathbf{Q})^k \cdot vec(\mathbf{I}_n).$$

Then, using tensor product properties $(\mathbf{Q} \otimes \mathbf{Q})^k = \mathbf{Q}^k \otimes \mathbf{Q}^k$ and $(\mathbf{Y} \otimes \mathbf{Z}) \cdot vec(\mathbf{I}_n) = vec(\mathbf{Z} \cdot \mathbf{I}_n \cdot \mathbf{Y}^T)$ with $\mathbf{Y} = \mathbf{Z} = \mathbf{Q}^k$, plus the linearity of $vec(\star)$, we can derive Eq.(7.1). □

The term $(1 - C)$ in Eq.(7.1) aims to normalize similarities in $[0, 1]$ as

$$\left\| \sum_{l=0}^{\infty} C^l \cdot \mathbf{Q}^l \cdot (\mathbf{Q}^T)^l \right\|_{\max} \leq \sum_{l=0}^{\infty} C^l = \frac{1}{1 - C}.$$ [5]

[5] The matrix norm $\|\mathbf{X}\|_{\max} = \max_{i,j} |[\mathbf{X}]_{i,j}|$ is the maximum absolute entry of $\mathbf{X}$.

Lemma 7.7 reformulates SimRank in the form of weight sum of all *symmetric* in-link paths of length $2l$ for node-pair (i, j). To clarify this, as $\mathbf{Q}$ is the weighted (*i.e.,* row-normalized) matrix of $\mathbf{A}^T$, Lemma 7.7 implies that $[\mathbf{Q}^l \cdot (\mathbf{Q}^T)^l]_{i,j}$ can tally *the weight sum* (instead of *the number*) of [6] in-link paths of length $2l$ for node-pair (i, j). Formally, we state this below:

Corollary 7.8. $[\mathbf{Q}^l \cdot (\mathbf{Q}^T)^l]_{i,j} = 0 \Leftrightarrow [(\mathbf{A}^T)^l \cdot \mathbf{A}^l]_{i,j} = 0.$ □

This, together with the component form of Eq.(7.1), *i.e.,*

$$[\mathbf{S}]_{i,j} = (1 - C) \cdot \sum_{l=0}^{\infty} C^l \cdot [\mathbf{Q}^l \cdot (\mathbf{Q}^T)^l]_{i,j}, \quad (\forall i, j \in [1, n]) \tag{7.2}$$

implies that $[\mathbf{S}]_{i,j}$ considers only contributions of *symmetric* in-link paths for (i, j), neglecting all *dissymmetric* ones. Consequently, $[\mathbf{S}]_{i,j} = 0$ if (i, j) has no symmetric paths. This proves the "zero-similarity" problem for SimRank.

Non-SimRank Based Metrics. Other measures, *e.g.,* Random Walk with Restart (RWR) and Personalized PageRank (PPR), also imply a SimRank-like "zero-similarity" issue.

As PPR is just a special vector form of RWR, our following discussion will mainly focus on RWR, which also suites PPR.

The "zero-similarity" issue for RWR, similar to SimRank, is that "nodes i and j are assessed as dissimilar $s_{\mathsf{rwr}}(i, j) = 0$ if there are no paths with one direction from i to j". For example in Figure 7.1, h and d are still dissimilar for RWR, as both $h \leftarrow e \leftarrow \boxed{a} \rightarrow d$ and $h \leftarrow e \leftarrow \boxed{a} \rightarrow b \rightarrow f \rightarrow d$ have two directions. However, $s_{\mathsf{rwr}}(a, f) \neq 0$ as there exists a path $\boxed{a} \rightarrow b \rightarrow f$ with one direction ($\rightarrow$) from a to f. Thus, both RWR and SimRank may encounter "zero-similarity" issues. Indeed, in the language of in-link paths, while SimRank considers only *symmetric* in-link paths (whose "source" node is in the center), RWR merely tallies *unidirectional* in-link paths (whose "source" node is at one end), both of which are in a biased way to assess similarity.

[6] Note that $[\mathbf{Q}^l \cdot (\mathbf{Q}^T)^l]_{i,j} \neq 0 \Leftrightarrow [(\mathbf{A}^T)^l \cdot \mathbf{A}^l]_{i,j} \neq 0$, thus $[\mathbf{Q}^l \cdot (\mathbf{Q}^T)^l]_{i,j}$ still characterizes the *existence* (but not the *number*) of in-link paths of length $2l$ for (i, j).

To further clarify the "zero-similarity" issue for RWR, we can convert its closed form $\mathbf{S} = (1 - C) \cdot (\mathbf{I}_n - C \cdot \mathbf{W})^{-1}$ [TFP06] into the power series form

$$[\mathbf{S}]_{i,j} = (1 - C) \cdot \sum_{k=0}^{\infty} C^k \cdot [\mathbf{W}^k]_{i,j}. \tag{7.3}$$

As $\mathbf{W}$ is a weighted (*i.e.,* row-normalized) matrix of $\mathbf{A}$, we have

$$[\mathbf{W}^k]_{i,j} = 0 \;\Leftrightarrow\; [\mathbf{A}^k]_{i,j} = 0.$$

Thus, by Lemma 7.5, the drawback of RWR is clear: $[\mathbf{S}]_{i,j}$ only tallies the weight sum of paths with one direction from i to j, yet totally ignores in-link paths whose "source" node is not at node i.

In a nutshell, RWR may not resolve "zero-similarity" issues for SimRank, and vice versa. As will be seen in Figure 7.3, all nodes in the family tree G should have some relevances. Although RWR considers "Father and Me being similar" that is neglected by SimRank, it ignores "Me and Cousin being similar" that is accommodated by SimRank. Besides, both RWR and SimRank neglect "Me and Uncle being similar". Worse still, RWR fails to produce *symmetric* similarity ($s(i, j) \neq s(j, i)$). Since there is no path directed from Me to Father, RWR alleges "Me and Father being dissimilar". These call for a unified measure for similarity assessment.

7.2.2 SimRank*: A Remedy for SimRank

The reinterpretation of SimRank provides a new possible remedy to its "zero-similarity" problem.

SimRank* (Geometric Series Form). Since SimRank (*resp.*RWR) loses all *dissymmetric* (*resp.non-unidirectional*) in-link paths for node-pair (i, j), our treatment aims to compensate $s(i, j)$ for such a loss, by accommodating all *dissymmetric* (*resp.non-unidirectional*) in-link paths. Precisely, by adding the terms $[\mathbf{Q}^{l_1} \cdot (\mathbf{Q}^T)^{l_2}]_{i,j}$, $\forall l_1 \neq l_2$ (*resp.*$\forall l_1 \neq 0$), with appropriate weights, into the series form of SimRank (*resp.*RWR), we can derive a new treatment as follows:

$$\hat{\mathbf{S}} = (1 - C) \cdot \sum_{l=0}^{\infty} \frac{C^l}{2^l} \cdot \sum_{\alpha=0}^{l} \binom{l}{\alpha} \cdot \mathbf{Q}^{\alpha} \cdot (\mathbf{Q}^T)^{l-\alpha}. \tag{7.4}$$

Here, $\binom{l}{\alpha}$ is *the binomial coefficient* defined as $\binom{l}{\alpha} = \frac{l!}{\alpha!(l-\alpha)!}$. We call Eq.(7.4) *the geometric* [7] *series form of SimRank**.

To see how the geometric form of SimRank* Eq.(7.4) is derived and why it can perfectly resolve the "zero-similarity" problem for SimRank and RWR, we rewrite Eq.(7.4) as

$$[\hat{\mathbf{S}}]_{i,j} = (1-C) \cdot \sum_{l=0}^{\infty} C^l \cdot [\hat{\mathbf{T}}_l]_{i,j} \quad \text{with} \tag{7.5}$$

$$[\hat{\mathbf{T}}_l]_{i,j} = \frac{1}{2^l} \cdot \sum_{\alpha=0}^{l} \binom{l}{\alpha} \cdot [\mathbf{Q}^{\alpha} \cdot (\mathbf{Q}^T)^{l-\alpha}]_{i,j}. \quad (\forall i,j \in [1,n])$$

Below, to avoid ambiguity, we use $\hat{\mathbf{S}}$ to denote the exact SimRank* in Eq.(7.4), and $\mathbf{S}$ the exact SimRank in Eq.(7.1).

Comparing Eq.(7.5) with Eq.(7.2), we see that for a fixed l, SimRank* $\hat{s}(i,j)$ uses $\sum_{\alpha=0}^{l} \binom{l}{\alpha} \cdot [\mathbf{Q}^{\alpha} \cdot (\mathbf{Q}^T)^{l-\alpha}]_{i,j}$ in $[\hat{\mathbf{T}}_l]_{i,j}$ to consider *all* in-link paths of length l for node-pair (i,j) in a comprehensive way, as opposed to SimRank $s(i,j)$ using $[\mathbf{Q}^l \cdot (\mathbf{Q}^T)^l]_{i,j}$ in Eq.(7.2) to accommodate only *symmetric* in-link paths of length $2l$ for node-pair (i,j) in a biased manner. As a result, SimRank* may find all (dissymmetric) in-link paths of two kinds, both of which are ignored by SimRank: (1) in-link paths of odd length; (2) in-link paths of even length whose in-link "source" is not in the center.

Though RWR via Eq.(7.3) using $[\mathbf{W}^l]_{i,j}$ may consider part of in-link paths of odd length that are missed by SimRank, they ignore (non-unidirectional) in-link paths of two kinds: (1) all symmetric ones that are accommodated by SimRank; (2) dissymmetric ones whose in-link "source" is not at an end, both of which can be found by SimRank*.

For instance, given a node-pair (i,j), Figure 7.2 compares all in-link paths of length $l \in [1,4]$ considered by SimRank, RWR, and SimRank*. It can be seen from 'SimRank* Column' that only a small number of in-link paths can be accommodated by SimRank (in dark gray cells) and RWR (in light gray cells), relative to those of SimRank*.

Weighted Factors of Two Types. We next elaborate on two kinds of weighted factors adopted by SimRank* Eq.(7.5): (1) *length weights* $\{C^l\}_{l=0}^{\infty}$, (2) *symmetry weights*

[7] Since $\{C^l\}$ in Eq.(7.4) is a *geometric* sequence, we abuse the term "*geometric*" for this series form, to distinguish Eq.(7.8).

Length	SimRank	RWR / PPR	α	SimRank*
1	N/A	$\boxed{i} \to j$	0	$\boxed{i} \to j$
			1	$i \leftarrow \boxed{j}$
2	$i \leftarrow \boxed{\bullet} \to j$	$\boxed{i} \to \circ \to j$	0	$\boxed{i} \to \circ \to j$
			1	$i \leftarrow \boxed{\bullet} \to j$
			2	$i \leftarrow \circ \leftarrow \boxed{j}$
3	N/A	$\boxed{i} \to \circ \to \circ \to j$	0	$\boxed{i} \to \circ \to \circ \to j$
			1	$i \leftarrow \boxed{\bullet} \to \circ \to j$
			2	$i \leftarrow \circ \leftarrow \boxed{\bullet} \to j$
			3	$i \leftarrow \circ \leftarrow \circ \leftarrow \boxed{j}$
4	$i \leftarrow \circ \leftarrow \boxed{\bullet} \to \circ \to j$	$\boxed{i} \to \circ \to \circ \to \circ \to j$	0	$\boxed{i} \to \circ \to \circ \to \circ \to j$
			1	$i \leftarrow \boxed{\bullet} \to \circ \to \circ \to j$
			2	$i \leftarrow \circ \leftarrow \boxed{\bullet} \to \circ \to j$
			3	$i \leftarrow \circ \leftarrow \circ \leftarrow \boxed{\bullet} \to j$
			4	$i \leftarrow \circ \leftarrow \circ \leftarrow \circ \leftarrow \boxed{j}$

$\circ$ – any node in $\mathcal{G}$ $\boxed{i}$, $\boxed{\bullet}$, $\boxed{j}$ – in-link "source"

Figure 7.2: In-link Paths of (i, j) for Length $l \in [1, 4]$ Counted by SimRank, RWR/PPR, and SimRank*

$\{\binom{l}{\alpha}\}_{\alpha=0}^{l}$.

Intuitively, *the length weight* C^l $(0 < C < 1)$ measures the importance of in-link paths of different lengths. Similar to the original SimRank (Eq.(7.2)), the outer summation over l in SimRank* (Eq.(7.5)) is to add up the contributions of in-paths of different length l. The length weight C^l aims at reducing the contributions of in-paths of *long* lengths relative to *short* ones, as $\{C^l\}_{l\in[0,\infty)}$ is a deceasing sequence *w.r.t.* length l.

The symmetry weight uses binomial $\binom{l}{\alpha}$ $(0 \le \alpha \le l)$ to assess the importance of in-link paths of a fixed length l, with α edges in one direction (from the "source" node to one end of the path) and $l - \alpha$ edges in the opposite direction. Here, α reflects the symmetry of in-link paths of length l. As depicted in Figure 7.2, when $\alpha = 0$ or l, in-link paths are totally dissymmetric, reducing to one single direction; when α is close to $\lfloor l/2 \rfloor$, the "source" node is near the center of in-link paths, being almost symmetric. To show the use of binomial $\binom{l}{\alpha}$ is reasonable, we consider the following issues.

(a) Why $\binom{l}{\alpha}$ is assigned only to $l+1$ kinds of *in-link paths*, for a fixed l? Say, for $l = 4$ in Figure 7.2, why neglect paths $\rho_1 : i \to \circ \leftarrow \circ \to \circ \leftarrow j$ and $\rho_2 : i \leftarrow \star \to \circ \leftarrow \diamond \to j$?

(b) Why use $\binom{l}{\alpha}$, instead of others, to weigh in-link paths?

(c) Why symmetric in-link paths are considered to be more important than less symmetric ones, for a fixed length?

For (a), as our SimRank* framework is *in-link oriented*,[8] the impact of *out-links* on similarity is not accommodated. Thus, for $l = 4$, the path ρ_1 is not considered since there are no *in-links* to nodes i and j in ρ_1. Even if i or j has in-links yet without *one common* in-link "source", *e.g.,* ρ_2, this path also has no contributions to similarity $\hat{s}(i, j)$. This is because in ρ_2 there are no in-links to nodes $\star$ and $\diamond$, thus the sub-path $\star \rightarrow \circ \leftarrow \diamond$ of ρ_2 has no contributions to $\hat{s}(\star, \diamond)$, which, iteratively, has no contributions to $\hat{s}(i, j)$. Hence, due to our *in-link oriented* framework for similarity assessment, for a fixed l, there are *at most* $l + 1$ kinds of *in-link paths* (where binomial weights $\binom{l}{\alpha}$ are assigned) having contributions to $\hat{s}(i, j)$, with $\alpha \in [0, l]$ edges in one direction and $l - \alpha$ edges in the opposite one, as shown in Figure 7.2.

For (b), there are 2 reasons for using $\binom{l}{\alpha}$ instead of others: (i) The binomial $\binom{l}{\alpha}$ can reduce the contributions of less symmetric in-link paths, relative to symmetric ones. Indeed, a larger (*resp.*smaller) weight is expected for an in-link path whose "source" is closer to the center (*resp.*either of ends). $\binom{l}{\alpha}$ happens to have this monotonicity: For a fixed l, when α increases from 0 to l, $\binom{l}{\alpha}$ first increases from 1 to a maximum value ($\alpha = \lfloor \frac{l}{2} \rfloor$, "source" at the center), and then "symmetrically" decreases back to 1 ($\alpha = l$, "source" at one end). (ii) The binomial $\binom{l}{\alpha}$ is an easy-to-compute math function, which enables the infinite series (Eq.(7.4)) to be simplified, as will be seen shortly, into the very succinct and elegant recurrence form (Eq.(7.10)). To our best knowledge, although some functions, like $e^{-(l-\frac{\alpha}{2})^2}$, have the similar monotonicity of $\binom{l}{\alpha}$, they would adversely complicate the form of Eq.(7.4) since it is even hard to compute $\sum_{\alpha=0}^{l} e^{-(l-\frac{\alpha}{2})^2}$ to determine the normalized weight factors, not to mention being able to simplify Eq.(7.4) into the elegant recurrence form. In contrast, $\sum_{\alpha=0}^{l} \binom{l}{\alpha} = 2^l$ enjoys a neat form. Inspired by these, we use $\binom{l}{\alpha}$, instead of others, as the preferred symmetric weight.

[8] In order to highlight the essence of "zero-SimRank" issue, our SimRank* model, just like SimRank,

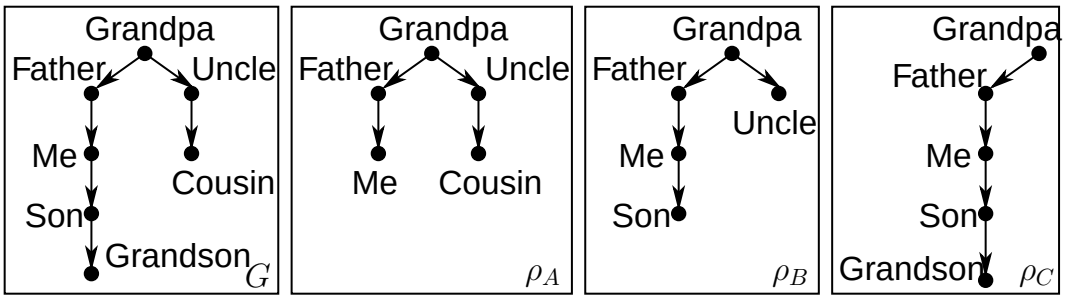


Figure 7.3: The more symmetric the in-link paths are, the larger contributions they will have to similarity

For (c), the example below can explain, for a fixed length, why larger weights are assigned to more symmetric paths. Consider paths ρ_A, ρ_B and ρ_C of a family tree in Figure 7.3. Most people might feel ρ_A (Me and Cousin being similar) is more reliable than ρ_B (Uncle and Son being similar), which is more reliable than ρ_C (Grandpa and Grandson being similar). Thus, the more symmetric the in-link path is, the larger contribution it has to similarity assessment. In Figure 7.3, ρ_A should have the largest weight, ρ_B the second, ρ_C the third.

The efficacy of $(1-C)$ and $\frac{1}{2^l}$ in Eq.(7.5) is to normalize $[\hat{\mathbf{S}}]_{i,j}$ and $[\hat{\mathbf{T}}_l]_{i,j}$, respectively, into $[0,1]$. More specifically, one can readily verify that $\|\mathbf{Q}^{l_1} \cdot (\mathbf{Q}^T)^{l_2}\|_{\max} \leq 1$, for $\forall l_1, l_2$. Thus, (i) $\|\sum_{\alpha=0}^{l} \binom{l}{\alpha} \cdot \mathbf{Q}^{\alpha} \cdot (\mathbf{Q}^T)^{l-\alpha}\|_{\max} \leq \sum_{\alpha=0}^{l} \binom{l}{\alpha} = 2^l$, which implies $\|\hat{\mathbf{T}}_l\|_{\max} \leq 1$. (ii) Since $\|\sum_{l=0}^{\infty} C^l \cdot \hat{\mathbf{T}}_l\|_{\max} \leq \sum_{l=0}^{\infty} C^l = \frac{1}{1-C}$, it follows that $\|\mathbf{S}\|_{\max} \leq 1$.

Combining these two kinds of weights, the contribution of any in-link path for a given node-pair can be easily assessed. For example in Figure 7.1, $h \leftarrow e \leftarrow \boxed{a} \rightarrow d$ has a contribution rate of $(1-0.8) \cdot 0.8^3 \frac{1}{2^3} \binom{3}{2} = 0.0384$ for node-pair (h, d), Similarly, $h \leftarrow e \leftarrow \boxed{a} \rightarrow b \rightarrow f \rightarrow d$ has a contribution rate of $(1-0.8) \cdot 0.8^5 \frac{1}{2^5} \binom{5}{2} = 0.0205$. As opposed to SimRank only using length weight C^l, SimRank* considers both C^l and symmetry weight $\binom{l}{\alpha}$. Thus, our revision resolves "zero-SimRank" issues, as well as inherits SimRank philosophy.

Convergence of SimRank*. As SimRank* in Eq.(7.4) is an *infinite* series, it is unclear whether this series is convergent. This motivates us to study its convergence issue.

PageRank, and RWR, is based on *incoming* edges for assessing similarity.

Let us first define *the k-th partial sum* of Eq.(7.4) as

$$\hat{\mathbf{S}}_k = (1-C)\cdot\sum_{l=0}^{k}\frac{C^l}{2^l}\cdot\sum_{\alpha=0}^{l}\binom{l}{\alpha}\cdot\mathbf{Q}^{\alpha}\cdot(\mathbf{Q}^T)^{l-\alpha}. \tag{7.6}$$

Leveraging $\hat{\mathbf{S}}_k$, we next show the convergence of Eq.(7.4).

Lemma 7.9. Let $\hat{\mathbf{S}}$ and $\hat{\mathbf{S}}_k$ be defined by Eqs.(7.4) and (7.6), respectively. Then, the gap between $\hat{\mathbf{S}}$ and $\hat{\mathbf{S}}_k$ is bounded by

$$\|\hat{\mathbf{S}}-\hat{\mathbf{S}}_k\|_{\max}\le C^{k+1}.\quad(\forall k=0,1,\cdots)\quad\square \tag{7.7}$$

PROOF. For each $k=0,1,\cdots$, we subtract Eq.(7.6) from Eq.(7.4), and then take $\|\star\|_{\max}$ norms on both sides to get

$$\begin{aligned}\|\hat{\mathbf{S}}-\hat{\mathbf{S}}_k\|_{\max} &\le (1-C)\sum_{l=k+1}^{\infty}\frac{C^l}{2^l}\cdot\underbrace{\sum_{\alpha=0}^{l}\binom{l}{\alpha}}_{=2^l}\cdot\underbrace{\|\mathbf{Q}^{\alpha}\cdot(\mathbf{Q}^T)^{l-\alpha}\|_{\max}}_{\le 1}\\ &\le (1-C)\sum_{l=k+1}^{\infty}C^l=(1-C)\cdot\frac{C^{k+1}}{(1-C)}=C^{k+1}.\quad\square\end{aligned}$$

The convergence of SimRank* (Eq.(7.4)) follows immediately from Lemma 7.9 and $\lim_{k\to\infty}C^{k+1}=0\ \ (0<C<1)$.

SimRank* (Exponential Series Form). In the *geometric* series form of SimRank* (Eq.(7.4)), Lemma 7.9 implies that, to guarantee the accuracy ϵ, the K-th partial sum $\hat{\mathbf{S}}_K$ with $K=\lceil\log_C\epsilon\rceil$ can be used to approximate the exact solution. However, there is a variant of SimRank* that can use only the K'-th partial sum with $K'\le K$ to ensure the same ϵ:

$$\hat{\mathbf{S}}'=e^{-C}\cdot\sum_{l=0}^{\infty}\frac{C^l}{l!}\cdot\frac{1}{2^l}\sum_{\alpha=0}^{l}\binom{l}{\alpha}\cdot\mathbf{Q}^{\alpha}\cdot(\mathbf{Q}^T)^{l-\alpha}. \tag{7.8}$$

We call Eq.(7.8) the *exponential series form of SimRank**. It differs from Eq.(7.4) in the length weight $\frac{C^l}{l!}$ (which is an *exponential* sequence *w.r.t.* l) and its normalized factor e^{-C}.

The exponential series form of SimRank* is introduced to improve the rate of convergence for similarity computation. To clarify this, we define $\hat{\mathbf{S}}'_k$ as the k-th partial sum of

$\hat{\mathbf{S}}'$ in Eq.(7.8). Analogous to Lemma 7.9, one can readily prove

$$\|\hat{\mathbf{S}}' - \hat{\mathbf{S}}'_k\|_{\max} \leq \frac{C^{k+1}}{(k+1)!}. \quad (\forall k = 0, 1, \cdots) \tag{7.9}$$

Comparing Eq.(7.9) with Eq.(7.7), we see that for any fixed k, as $\frac{C^{k+1}}{(k+1)!} \leq C^{k+1}$, the convergence rate of $\hat{\mathbf{S}}'_k$ is *always* faster than that of $\hat{\mathbf{S}}_k$. Hence, to guarantee the same accuracy, the exponential SimRank* only needs to compute a tiny fraction of the partial sums of the geometric SimRank*.

The choice of length weight $\frac{C^l}{l!}$ for the exponential SimRank* (Eq.(7.8)) plays a key role in accelerating convergence. As suggested by the proof of Lemma 7.9, the bound C^{k+1} in Eq.(7.7) (*resp.* $\frac{C^{k+1}}{(k+1)!}$ in Eq.(7.9)) is actually derived from our choice of length weight C^l (*resp.* $\frac{C^l}{l!}$) for the geometric (*resp.*exponential) SimRank*. Thus, there might exist other length weights for speeding up the convergence of SimRank*, as there is no sanctity of the earlier choices of length weight. That is, apart from C^l and $\frac{C^l}{l!}$, other sequence, *e.g.,* $\frac{C^l}{l}$, that satisfies decreasing monotonicity *w.r.t.* length l can be regarded as another possible candidate for length weight, since the efficacy of the length weight is to reduce the contributions of in-link paths of long lengths relative to short ones. The reasons why we select C^l and $\frac{C^l}{l!}$, instead of others, are two-fold: (1) The normalized factor of length weight should have a simple form, *e.g.,* $\sum_{l=0}^{\infty} \frac{C^l}{l!} = e^C$. (2) Once selected, the length weight should enable the series form of SimRank* to be simplified into a very elegant form, *e.g.,* using $\frac{C^l}{l!}$ allows Eq.(7.8) being simplified, as will be seen in Eq.(7.12), into a neat closed form. In contrast, $\frac{C^l}{l}$ is not a preferred length weight as its series version may not be simplified into a neat recursive (or closed) form, though the form $\sum_{l=0}^{\infty} \frac{C^l}{l} = \ln \frac{1}{(1-C)}$ is simple for normalized factor.

7.3 Efficiently Computing SimRank*

At first glance, the series form of SimRank* (Eq.(7.4)) is more complicated than that of SimRank (Eq.(7.1)). A brute-force way of computing the first k-th partial sums of Eq.(7.4) requires $O(k \cdot l^2 \cdot n^3)$ time, involving l^2 matrix multiplications in the inner

summation for each fixed l in the outer summation, which seems much more expensive than SimRank.

In this section, we first reformulate the series forms of SimRank* into elegant recursive and closed forms. We then propose efficient techniques for computing SimRank*.

7.3.1 Recursive & Closed Forms of SimRank*

The series forms of SimRank* (Eqs.(7.4) and (7.8)) are tedious, and suffer from high complexity if calculated directly.

The main result of this subsection is to derive an elegant recursive form for Eq.(7.4) and a closed form for Eq.(7.8), which will be useful for efficient SimRank* computation.

Recursive Form of Geometric SimRank*. We first show a recursive form for the geometric SimRank* of Eq.(7.4).

Theorem 7.10. *The SimRank* geometric series* $\hat{\mathbf{S}}$ *in Eq.(7.4) takes the following elegant recursive form:*

$$\hat{\mathbf{S}} = \tfrac{C}{2} \cdot (\mathbf{Q} \cdot \hat{\mathbf{S}} + \hat{\mathbf{S}} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n. \quad \square \tag{7.10}$$

To prove Theorem 7.10, the following lemma is needed.

Lemma 7.11. For each $k = 0, 1, \cdots$, the k-th partial sum $\hat{\mathbf{S}}_k$ defined by Eq.(7.6) satisfies the following iteration:

$$\begin{cases} \hat{\mathbf{S}}_0 = (1 - C) \cdot \mathbf{I}_n, \\ \hat{\mathbf{S}}_{k+1} = \tfrac{C}{2} \cdot (\mathbf{Q} \cdot \hat{\mathbf{S}}_k + \hat{\mathbf{S}}_k \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n. \end{cases} \quad \square \tag{7.11}$$

Proof. For $k = 0$, it is obvious from Eq.(7.6) that $\hat{\mathbf{S}}_0 = (1 - C) \cdot \mathbf{I}_n$, which satisfies Eq.(7.11). For $k = 1, 2, \cdots$, substituting Eq.(7.6) into the right-hand side of Eq.(7.11) yields

$$\hat{\mathbf{S}}_{k+1} = \frac{C}{2} \cdot \Bigg((1 - C) \sum_{l=0}^{k} \frac{C^l}{2^l} \overbrace{\sum_{\alpha=0}^{l} \binom{l}{\alpha} \cdot \mathbf{Q}^{\alpha+1} \cdot (\mathbf{Q}^T)^{l-\alpha}}^{=\sum_{\alpha=1}^{l+1} \binom{l}{\alpha-1} \cdot \mathbf{Q}^{\alpha} \cdot (\mathbf{Q}^T)^{l-\alpha+1}} + \\ + (1 - C) \sum_{l=0}^{k} \frac{C^l}{2^l} \sum_{\alpha=0}^{l} \binom{l}{\alpha} \cdot \mathbf{Q}^{\alpha} \cdot (\mathbf{Q}^T)^{l-\alpha+1} \Bigg) + (1 - C) \cdot \mathbf{I}_n$$

$$
\begin{aligned}
&= \frac{C}{2}(1-C)\left(\sum_{l=0}^{k}\frac{C^l}{2^l}\left(\sum_{\alpha=1}^{l}\left(\binom{l}{\alpha-1}+\binom{l}{\alpha}\right)\mathbf{Q}^{\alpha}(\mathbf{Q}^T)^{l-\alpha+1}\right)+\right.\\
&\quad \left.+\mathbf{Q}^{l+1}+(\mathbf{Q}^T)^{l+1}\right)+(1-C)\cdot\mathbf{I}_n\\
&= \frac{C}{2}\cdot(1-C)\left(\sum_{l=0}^{k}\frac{C^l}{2^l}\cdot\sum_{\alpha=0}^{l+1}\binom{l+1}{\alpha}\cdot\mathbf{Q}^{\alpha}(\mathbf{Q}^T)^{l-\alpha+1}\right)+(1-C)\mathbf{I}_n\\
&= (1-C)\cdot\sum_{l=0}^{k+1}\frac{C^l}{2^l}\cdot\sum_{\alpha=0}^{l}\binom{l}{\alpha}\cdot\mathbf{Q}^{\alpha}\cdot(\mathbf{Q}^T)^{l-\alpha}.
\end{aligned}
$$

Thus, $\hat{\mathbf{S}}_{k+1}$ in Eq.(7.11) also takes the form of Eq.(7.6). □

One consequence of Lemma 7.11 is the proof of Theorem 7.10.

Proof of Theorem 7.10. Lemma 7.9 implies the convergence of SimRank*, *i.e.,* the existence of $\lim_{k\to\infty}\hat{\mathbf{S}}_k$. Thus, taking limits on both sides of Eq.(7.11) as $k\to\infty$ yields Eq.(7.10). □

Closed Form of Exponential SimRank*. We next present a closed formula for the exponential SimRank* of Eq.(7.8).

Theorem 7.12. *The exponential series form of SimRank* in Eq.(7.8) neatly takes the following closed form:*

$$\hat{\mathbf{S}}' = e^{-C}\cdot e^{\frac{C}{2}\mathbf{Q}}\cdot e^{\frac{C}{2}\mathbf{Q}^T}.\ ^{9} \qquad \square \tag{7.12}$$

Proof. We utilize the factorial formula $\binom{l}{\alpha}=\frac{l!}{\alpha!(l-\alpha)!}$ to simplify the series form of Eq.(7.8) into the closed form:

$$
\begin{aligned}
\hat{\mathbf{S}}' &= e^{-C}\cdot\sum_{l=0}^{\infty}\sum_{\alpha=0}^{l}\frac{1}{2^l}\cdot\frac{C^{\alpha}}{\alpha!}\mathbf{Q}^{\alpha}\cdot\frac{C^{l-\alpha}}{(l-\alpha)!}(\mathbf{Q}^T)^{l-\alpha}\\
&= e^{-C}\cdot\sum_{\alpha=0}^{\infty}\frac{C^{\alpha}}{\alpha!}\mathbf{Q}^{\alpha}\cdot\underbrace{\sum_{l=\alpha}^{\infty}\frac{1}{2^l}\cdot\frac{C^{l-\alpha}}{(l-\alpha)!}(\mathbf{Q}^T)^{l-\alpha}}_{=\sum_{l=0}^{\infty}\frac{1}{2^{l+\alpha}}\cdot\frac{t^l}{l!}(\mathbf{Q}^T)^l}\\
&= e^{-C}\cdot\left(\sum_{\alpha=0}^{\infty}\frac{1}{2^{\alpha}}\cdot\frac{C^{\alpha}}{\alpha!}\mathbf{Q}^{\alpha}\right)\cdot\left(\sum_{l=0}^{\infty}\frac{1}{2^l}\cdot\frac{C^l}{l!}(\mathbf{Q}^T)^l\right)
\end{aligned}
$$

[9] $e^{\mathbf{X}}\triangleq\mathbf{I}+\mathbf{X}+\frac{\mathbf{X}^2}{2!}+\cdots=\sum_{k=0}^{\infty}\frac{\mathbf{X}^k}{k!}$, for a square matrix $\mathbf{X}$.

$$= e^{-C} \cdot e^{\frac{C}{2}\mathbf{Q}} \cdot e^{\frac{C}{2}\mathbf{Q}^T},$$

where the second equality is obtained by interchanging the order of double summation $\sum_{l=0}^{\infty}\sum_{\alpha=0}^{l} f(l,\alpha) = \sum_{\alpha=0}^{\infty}\sum_{l=\alpha}^{\infty} f(l,\alpha)$. □

The utility of Theorem 7.12 will be appreciated in Subsect. 7.3.3 for optimizing the exponential SimRank* computation.

7.3.2 SimRank* Computation

Having formulated SimRank* into the very elegant forms, we next develop efficient techniques to speed up the computation of SimRank*.

Due to high commonalities between the geometric SimRank* $\hat{\mathbf{S}}$ (in Eq.(7.4)) and its exponential variant $\hat{\mathbf{S}}'$ (in Eq.(7.8)), we shall mainly focus on geometric SimRank* computation, which is readily applicable to its exponential variant as well.

Algorithm. To compute the SimRank* series $\hat{\mathbf{S}}$ in Eq.(7.4), the closed form Eq.(7.10) provides an easy yet effective way: One can use the iterative paradigm Eq.(7.11) to compute $\hat{\mathbf{S}}_k$, with accuracy guaranteed by Lemma 7.9.

Complexity. The computational time of performing Eq.(7.11) is $O(Knm)$ for K iterations on a graph of n nodes and m edges, which is dominated by the cost of matrix multiplication $\mathbf{Q} \cdot \hat{\mathbf{S}}_k$ per iteration. Due to $\hat{\mathbf{S}}_k$ symmetry, the result of $\hat{\mathbf{S}}_k \cdot \mathbf{Q}^T$ can be obtained from the transpose of the calculated matrix $\mathbf{Q} \cdot \hat{\mathbf{S}}_k$. Thus, for each iteration, Eq.(7.11) requires only one matrix multiplication (corresponding to performing only a *single* summation of Eq.(7.11)), as opposed to its counterpart of computing SimRank via Eq.(2.2) that needs two matrix multiplications for $\mathbf{Q} \cdot \mathbf{S}_k \cdot \mathbf{Q}^T$ (corresponding to performing a *double* summation of Eq.(2.1) regardless of whether memoization [LVGT08] is used). From this perspective, despite the traversal of more in-link paths, SimRank* runs even faster (up to a constant factor) than SimRank, which is a substantial improvement achieved by Theorem 7.10.

7.3.3 Optimizations

To accelerate SimRank* iterations in Eq.(7.11), the conventional optimization techniques [LVGT08] for SimRank cannot be effectively applied to SimRank*. Indeed, Lizorkin *et al.* [LVGT08] proposed three appealing approaches for optimizing SimRank computation, *i.e.,* essential node-pair selection, partial sums memoization, and threshold-sieved similarities, among which only the threshold-sieved similarities method can be ported to SimRank* that allows eliminating node-pairs of small similarities in the computation. Essential node-pair selection no longer applies because SimRank utilizes a "zero-similarity" set as a pruning rule to speed up its computation, whereas SimRank* regards the existence of such a set as an issue of the SimRank philosophy and attempts to fix it. Partial sums memoization plays a vital role in significantly speeding up the computation of SimRank to $O(Knm)$ time. To see why it does not work in SimRank*, let us compare the component forms of SimRank and SimRank*, respectively, in Eqs.(7.13) and (7.14):

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{x\in\mathcal{I}(a)} \overbrace{\sum_{y\in\mathcal{I}(b)} s_k(x,y)}^{=\text{Partial}^{s_k}_{\mathcal{I}(b)}(x)}. \tag{7.13}$$

$$\hat{s}_{k+1}(a,b) = \frac{C}{2\,|\mathcal{I}(b)|} \underbrace{\sum_{y\in\mathcal{I}(b)} \hat{s}_k(a,y)}_{=\text{Partial}^{\hat{s}_k}_{\mathcal{I}(b)}(a)} + \frac{C}{2\,|\mathcal{I}(a)|} \sum_{x\in\mathcal{I}(a)} \hat{s}_k(x,b). \tag{7.14}$$

For SimRank, if $\mathcal{I}(a)$ and $\mathcal{I}(\star)$ have some node, say i, in common, then the partial sum $\text{Partial}^{s_k}_{\mathcal{I}(b)}(i)$ in Eq.(7.13), once memoized, can be reused in both $\hat{s}_{k+1}(a,b)$ and $\hat{s}_{k+1}(\star,b)$ computation. In contrast, for SimRank*, no matter whether $\mathcal{I}(a)\cap\mathcal{I}(\star) \neq \varnothing$, the partial sum $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(b)}(a)$ in Eq.(7.14) for computing $\hat{s}_{k+1}(a,b)$, if memoized, has no chance to be reused again in computing other similarities $\hat{s}_{k+1}(\star,b)$, with $\star$ denoting any node in $\mathcal{G}$ except a.

Fine-grained Memoization. Instead of memoizing the results of $\sum_{y\in\mathcal{I}(b)} \hat{s}_k(a,y)$ over the *whole* set $\mathcal{I}(b)$ in Eq.(7.14), we use *fine-grained* memoization for optimizing SimRank*

by caching a partial sum, in part, over a *subset* as follows:

$$\text{Partial}_{\Delta}^{\hat{s}_k}(a) \triangleq \sum_{y \in \Delta} \hat{s}_k(a, y) \text{ with } \Delta \subseteq \mathcal{I}(\star).$$

Our observation is that there may be duplicate additions among $\sum_{y \in \mathcal{I}(\star)} \hat{s}_k(a, y)$ over *different* in-neighbor sets $\mathcal{I}(\star)$. Thus, once memoized, the result of $\text{Partial}_{\Delta}^{\hat{s}_k}(a)$ can be shared among many sums $\sum_{y \in \mathcal{I}(\star)} \hat{s}_k(a, y)$ for computing $\hat{s}_{k+1}(a, \star)$. As an example in Figure 7.1, $\mathcal{I}(h)$ and $\mathcal{I}(i)$ have three nodes $\{e, j, k\}$ in common, and thus, once memoized, the resulting fine-grained partial sum $\text{Partial}_{\{e,j,k\}}^{\hat{s}_k}(a)$ can be shared between $\sum_{y \in \mathcal{I}(h)} \hat{s}_k(a, y)$ and $\sum_{y \in \mathcal{I}(i)} \hat{s}_k(a, y)$ for computing both $\hat{s}_{k+1}(a, h)$ and $\hat{s}_{k+1}(a, i)$ via Eq.(7.14), for any fixed a. However, it seems hard to find perfect fine-grained subsets $\Delta \subseteq \mathcal{I}(\star)$ for maximal computation sharing, since there may be many arbitrarily overlapped in-neighbor sets in a graph. To overcome this difficulty, we shall deploy efficient techniques of bipartite graph compression via edge concentration for finding such fine-grained subsets.

Induced Bigraph. We first construct *an induced bipartite graph (bigraph)* from $\mathcal{G}$, which is defined as follows.

Definition 7.13. An induced bipartite graph (bigraph) from a given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a bipartite graph $\tilde{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B}, \tilde{\mathcal{E}})$, such that its two disjoint node sets $\mathcal{T} = \{x \in \mathcal{V} \mid \mathcal{O}(x) \neq \varnothing\}$, $\mathcal{B} = \{x \in \mathcal{V} \mid \mathcal{I}(x) \neq \varnothing\}$, [10] and for each $u \in \mathcal{T}$ and $v \in \mathcal{B}$, $(u, v) \in \tilde{\mathcal{E}}$ if and only if there is an edge from u to v in $\mathcal{G}$. □

Intuitively, an induced bigraph $\tilde{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B}, \tilde{\mathcal{E}})$ visualizes the neighborhood structure of $\mathcal{G}$ from a different perspective. For any $x \in \mathcal{B}$, the nodes in $\mathcal{T}$ that are connected with x correspond to the in-neighbors of x in $\mathcal{G}$. Note that when node x has both in- and out-neighbors in $\mathcal{G}$, label x that appears in both $\mathcal{T}$ and $\mathcal{B}$ will be regarded as two distinct nodes despite the same label. To avoid ambiguity, we shall use $x \in \mathcal{T}$ and $x \in \mathcal{B}$ to distinguish them. Each directed edge in $\mathcal{G}$ is mapped to one edge in $\tilde{\mathcal{G}}$, and thus, $|\mathcal{E}| = |\tilde{\mathcal{E}}|$. For instance, the left part of Figure 7.4 shows the induced bigraph $\tilde{\mathcal{G}}$ from $\mathcal{G}$

[10] The notation $\mathcal{O}(x)$ denotes the out-neighbor set of node x.

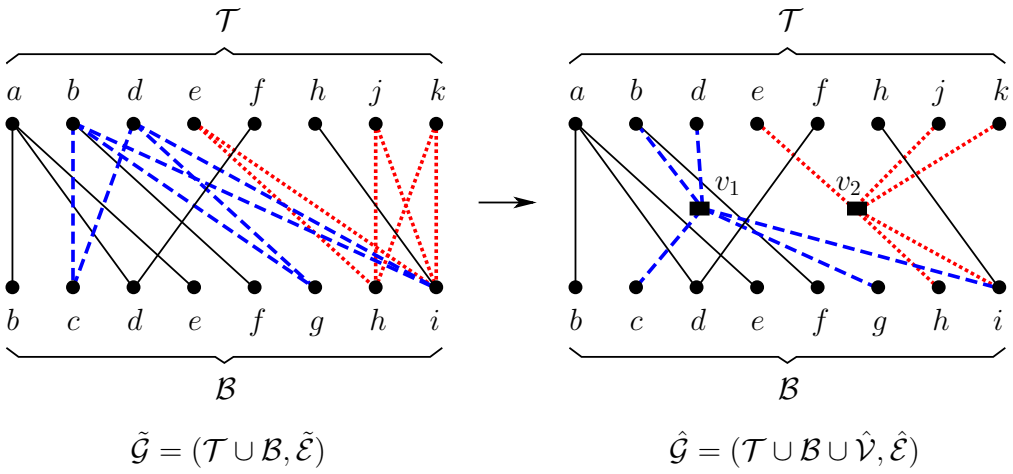


Figure 7.4: Compression of Induced Bigraph $\tilde{\mathcal{G}}$ into $\hat{\mathcal{G}}$ via Edge Concentration

of Figure 7.1. From $\tilde{\mathcal{G}}$, we can clearly see that b and d in $\mathcal{B}$ are both connected with a in $\mathcal{T}$, meaning that, in $\mathcal{G}$, b and d both have an in-neighbor a.

Biclique Compression via Edge Concentration. Based on the induced bigraph $\tilde{\mathcal{G}}$, we next introduce the notion of *bipartite cliques (bicliques)*.

Definition 7.14. Given an induced bigraph $\tilde{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B}, \tilde{\mathcal{E}})$, a pair of two disjoint subsets $\mathcal{X} \subseteq \mathcal{T}$ and $\mathcal{Y} \subseteq \mathcal{B}$ is called a biclique if $(x, y) \in \tilde{\mathcal{E}}$ for all $x \in \mathcal{X}$ and $y \in \mathcal{Y}$. □

Intuitively, a biclique $(\mathcal{X}, \mathcal{Y})$ is a complete bipartite subgraph of $\tilde{\mathcal{G}}$, which has $|\mathcal{X}|+|\mathcal{Y}|$ nodes and $|\mathcal{X}| \times |\mathcal{Y}|$ edges. Each biclique $(\mathcal{X}, \mathcal{Y})$ in $\tilde{\mathcal{G}}$ tells us that in $\mathcal{G}$, all nodes $y \in \mathcal{Y}$ have the common in-neighbor set $\mathcal{X}$. For example, there are two bicliques in Figure 7.4: $(\{b, d\}, \{c, g, i\})$ in dashed line, and $(\{e, j, k\}, \{h, i\})$ in dotted line. Biclique $(\{b, d\}, \{c, g, i\})$ in $\tilde{\mathcal{G}}$ implies that in $\mathcal{G}$, three nodes c, g, i all have two in-neighbors $\{b, d\}$ in common.

Bicliques are introduced to compress bigraph $\tilde{\mathcal{G}}$ for optimizing SimRank* computation. It is important to notice that for any fixed node a, the total cost [11] of performing the sums $\sum_{y \in \mathcal{I}(\star)} \hat{s}_k(a, y)$ over all in-neighbor sets $\mathcal{I}(\star)$ (via Eq.(7.14)) is equal to the

[11] Here, the total cost refers to the number of additions plus assignment operations. For example, the cost of performing $\sum_{y \in \mathcal{I}(h)} \hat{s}_k(a, y) = \hat{s}_k(a, e) + \hat{s}_k(a, j) + \hat{s}_k(a, k)$ is 3, including 2 additions and 1 assignment operation to store the result, which is equal to the number of edges that are connected with node $h \in \mathcal{B}$ in the left part of Figure 7.4.

number $|\tilde{\mathcal{E}}|$ of edges of bigraph $\tilde{\mathcal{G}}$. Therefore, our goal of minimizing the cost of summations for SimRank* is equivalent to the problem of minimizing the number of edges in the compressed graph of $\tilde{\mathcal{G}}$. Unfortunately, this bigraph compression problem, also known as *edge concentration* (EC), has been proved to be NP-hard [Lin00]. The main ingredient of EC is to group sets of edges in $\tilde{\mathcal{G}}$ together, so that the compressed graph contains fewer edges which often implies less cost of summations for SimRank*, while retaining the same information as $\tilde{\mathcal{G}}$. To compress $\tilde{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B}, \tilde{\mathcal{E}})$, we first leverage Buehrer and Chellapilla's algorithm [BC08b] for finding collections of bicliques in $\tilde{\mathcal{G}}$. Their algorithm is based on the heuristic of frequent itemset mining, and requires $O(|\tilde{\mathcal{E}}| \log(|\mathcal{T}| + |\mathcal{B}|))$ time to identify bicliques. We then replace edges of each biclique $(\mathcal{X}, \mathcal{Y})$ with a special node, called *an edge concentration node*, whose "fan-in" is all nodes in $\mathcal{X}$ and whose "fan-out" is all nodes in $\mathcal{Y}$. Finally, the compressed graph, denoted as $\hat{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B} \cup \hat{\mathcal{V}}, \hat{\mathcal{E}})$, can be obtained from bigraph $\tilde{\mathcal{G}}$, where (i) $\mathcal{T}$ and $\mathcal{B}$ are the same as those of $\tilde{\mathcal{G}}$, (ii) $\hat{\mathcal{V}}$ is the set of edge concentration nodes, and (iii) $\hat{\mathcal{E}}$ is the set of edges in $\hat{\mathcal{G}}$. In practice, $|\hat{\mathcal{E}}|$ is typically much smaller than $|\tilde{\mathcal{E}}|$, since $|\mathcal{X}| \times |\mathcal{Y}|$ edges of each biclique in $\tilde{\mathcal{G}}$ are reduced to $|\mathcal{X}| + |\mathcal{Y}|$ edges in $\hat{\mathcal{G}}$, which is a substantial improvement achieved by edge concentration. For example, the right part of Figure 7.4 depicts the resultant graph $\hat{\mathcal{G}}$ of applying this approach to $\tilde{\mathcal{G}}$. We can see that the number of edges in $\hat{\mathcal{G}}$ is decreased by 2 via edge concentration, meaning that the cost of computing SimRank* in $\hat{\mathcal{G}}$ can be reduced by 2 operations, by adding two edge concentration nodes v_1 and v_2.

Algorithm. Based on compressed graph $\hat{\mathcal{G}}$, we next present an algorithm for computing SimRank*, by using fine-grained memoization. The algorithm, referred to as memo-gSR*, is shown in Algorithm 7.1. It takes as input a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, a damping factor C, and the number of iterations K, and returns all-pairs of SimRank* similarities $\hat{s}(\star, \star)$.

To present the algorithm, we need the following notations. For a compressed graph $\hat{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B} \cup \hat{\mathcal{V}}, \hat{\mathcal{E}})$, we shall abuse (i) $\Delta(v)$ $(v \in \hat{\mathcal{V}})$ to denote all the "fan-in" nodes of concentration node v in $\mathcal{T}$, *i.e.,* $\Delta(v) = \{x \in \mathcal{T} \mid \exists (x, v) \in \hat{\mathcal{E}}\}$; and (ii) $\mathcal{N}(x)$ $(x \in \mathcal{B})$, to denote all the nodes in $\mathcal{T} \cup \hat{\mathcal{V}}$ that are connected with $x \in \mathcal{B}$, *i.e.,* $\mathcal{N}(x) = \{y \in$

Algorithm 7.1: memo-gSR* $(\mathcal{G}, C, K)$

Input : graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, damping factor C, iteration K.

Output: SimRank* scores $\hat{s}_K(\star, \star)$.

1 build an induced bigraph $\tilde{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B}, \tilde{\mathcal{E}})$ from $\mathcal{G} = (\mathcal{V}, \mathcal{E})$;

2 generate a compressed graph $\hat{\mathcal{G}} = (\mathcal{T} \cup \mathcal{B} \cup \hat{\mathcal{V}}, \hat{\mathcal{E}})$ from $\tilde{\mathcal{G}}$;

3 initialize $\hat{s}_0(x, y) \leftarrow \begin{cases} 1, & x=y \\ 0, & x \neq y \end{cases} \quad \forall x, y \in \mathcal{V}$;

4 **for** $k \leftarrow 0, 1, \cdots, K-1$ **do**

5 **foreach** *node* $v \in \hat{\mathcal{V}}$ *in* $\hat{\mathcal{G}}$ **do**

6 **foreach** *node* $a \in \mathcal{V}$ *in* $\mathcal{G}$ **do**

7 $\text{Partial}^{\hat{s}_k}_{\Delta(v)}(a) \leftarrow \sum_{y \in \Delta(v)} \hat{s}_k(a, y)$;

8 **foreach** *node* $x \in \hat{\mathcal{B}}$ *in* $\hat{\mathcal{G}}$ **do**

9 **foreach** *node* $a \in \mathcal{V}$ *in* $\mathcal{G}$ **do**

10 $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(x)}(a) \leftarrow \sum_{y \in \mathcal{N}(x) \cap \mathcal{T}} \hat{s}_k(a, y) + \sum_{y \in \mathcal{N}(x) \cap \hat{\mathcal{V}}} \text{Partial}^{\hat{s}_k}_{\Delta(y)}(a)$;

11 free $\text{Partial}^{\hat{s}_k}_{\Delta(v)}(a) \quad \forall v \in \hat{\mathcal{V}}, a \in \mathcal{V}$;

12 **foreach** *node* $x \in \mathcal{V}$ *in* $\mathcal{G}$ **do**

13 **foreach** *node* $y \in \mathcal{V}$ *in* $\mathcal{G}$ **do**

14 initialize $t_1 \leftarrow 0$, $t_2 \leftarrow 0$;

15 **if** $\mathcal{I}(x) \neq \varnothing$ **then** $t_1 \leftarrow \frac{C}{2|\mathcal{I}(x)|} \text{Partial}^{\hat{s}_k}_{\mathcal{I}(x)}(y)$;

16 **if** $\mathcal{I}(y) \neq \varnothing$ **then** $t_2 \leftarrow \frac{C}{2|\mathcal{I}(y)|} \text{Partial}^{\hat{s}_k}_{\mathcal{I}(y)}(x)$;

17 compute $\hat{s}_{k+1}(x, y) \leftarrow t_1 + t_2 + \begin{cases} 1-C, & x=y \\ 0, & x \neq y \end{cases}$;

18 free $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(x)}(y) \quad \forall x \in \mathcal{V}, y \in \mathcal{V}$;

19 **return** $\hat{s}_K(\star, \star)$;

$\mathcal{T} \cup \hat{\mathcal{V}} \mid \exists (y, x) \in \hat{\mathcal{E}}\}$.

The algorithm memo-gSR* runs in two phases.

(1) Preprocessing (lines 1–2). The algorithm first constructs an induced bigraph $\tilde{\mathcal{G}}$

from $\mathcal{G}$ (line 1). Based on $\mathcal{G}$, it then compresses $\tilde{\mathcal{G}}$ into $\hat{\mathcal{G}}$, by invoking the algorithm [BC08b] to replace bicliques of $\tilde{\mathcal{G}}$ with "stars" via edge concentration (line 2).

(2) Updating (lines 3–19). The algorithm then iteratively computes all $\hat{s}_k(\star,\star)$ based on $\hat{\mathcal{G}}$. For every iteration k, (i) it first uses fine-grained memoization to add up $\hat{s}_k(a,\star)$ for each fixed a, with $\star$ being each node in the "fan-in" set $\Delta(v)$ of an edge concentration node v in $\hat{\mathcal{V}}$ (lines 5–7). (ii) Using the memoized $\text{Partial}^{\hat{s}_k}_{\Delta(\star)}(a)$, it then computes the partial sums $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(\star)}(a)$ over different in-neighbor set $\mathcal{I}(\star)$ of $\mathcal{G}$ (lines 8–10). Due to fine-grained memoization, the intermediate results $\text{Partial}^{\hat{s}_k}_{\Delta(\star)}(a)$, for any fixed node a, can be reused among many partial sums $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(\star)}(a)$ computations. Hence, the cost of computing SimRank* is reduced. (iii) By Eq.(7.14), these partial sums can be used for computing $\hat{s}_k(\star,\star)$ (lines 12–17). Due to $\hat{s}_k(\star,\star)$ symmetry, the second summation in Eq.(7.14) can be computed from $\text{Partial}^{\hat{s}_k}_{\Delta(a)}(x)$ (line 16). Once processed, the memoized results are freed (lines 11 and 18). After K iterations, SimRank* scores $\hat{s}_K(\star,\star)$ of all-pairs are returned (line 19).

Correctness & Complexity. One can verify that the algorithm correctly computes $\hat{s}_K(\star,\star)$, which satisfies Eq.(7.14). Besides, memo-gSR* is in $O(Kn\tilde{m})$ time, where $\tilde{m}$ is the number of edges in the compressed graph $\hat{\mathcal{G}}$. Here, $\tilde{m}$ is *always* smaller than m, and in practice, $\tilde{m} \ll m$, depending on the number of bicliques, and biclique density in $\tilde{\mathcal{G}}$. This is because edge concentration compresses bicliques (dense subgraphs) in $\tilde{\mathcal{G}}$ such that for each biclique $(\mathcal{X}_i, \mathcal{Y}_i)$, the number of its edges $|\mathcal{X}_i| \cdot |\mathcal{Y}_i|$ can be reduced to $|\mathcal{X}_i| + |\mathcal{Y}_i|$. Thus, $\tilde{m} \leq m - N \cdot \sum_{i=1}^{N} (|\mathcal{X}_i| \cdot |\mathcal{Y}_i| - (|\mathcal{X}_i| + |\mathcal{Y}_i|))$, where N is the number of bicliques in $\tilde{\mathcal{G}}$. Since $|\mathcal{X}_i|, |\mathcal{Y}_i| \geq 2$, it holds that $|\mathcal{X}_i| + |\mathcal{Y}_i| \leq |\mathcal{X}_i| \cdot |\mathcal{Y}_i|$. Hence, $\tilde{m}$ is *always* less than m. Moreover, the construction of $\hat{\mathcal{G}}$ is in $O(|\tilde{\mathcal{E}}| \log(|\mathcal{T}| + |\mathcal{B}|)) = O(\tilde{m} \log(2n))$ time [BC08b] (lines 1–2). For each iteration, $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(\star)}(\star)$ are in $O(n\tilde{m})$ time (lines 5–11), and $\hat{s}_k(\star,\star)$ in $O(n^2)$ time (lines 12–18). Thus, the total time is $O(Kn\tilde{m})$, as opposed to $O(Knm)$ of original iterations in Lemma 7.11.

Example 7.15. Recall the graph $\mathcal{G}$ of Figure 7.1. memo-gSR* computes SimRank* scores of all-pairs in $\mathcal{G}$ as follows:

First, using Definition 7.13 and the algorithm [BC08b], it builds bigraph $\tilde{\mathcal{G}}$ and compressed graph $\hat{\mathcal{G}}$, as shown in Figure 7.4.

Then, it iteratively computes SimRank* via fine-grained memoization based on $\hat{\mathcal{G}}$. For example, to compute $\hat{s}_{k+1}(a,i)$ and $\hat{s}_{k+1}(a,h)$, it first memoizes the fine-grained partial sums over the "fan-in" sets of v_1 and v_2 (lines 5–7):

$$\text{Partial}^{\hat{s}_k}_{\Delta(v_1)}(a) \leftarrow \hat{s}(b,a) + \hat{s}(d,a),$$

$$\text{Partial}^{\hat{s}_k}_{\Delta(v_2)}(a) \leftarrow \hat{s}(e,a) + \hat{s}(j,a) + \hat{s}(k,a).$$

Using memoized $\text{Partial}^{\hat{s}_k}_{\Delta(v_1)}(a)$ and $\text{Partial}^{\hat{s}_k}_{\Delta(v_2)}(a)$, it then computes $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(i)}(a)$ and $\text{Partial}^{\hat{s}_k}_{\mathcal{I}(h)}(a)$ (lines 8–10)

$$\text{Partial}^{\hat{s}_k}_{\mathcal{I}(i)}(a) \leftarrow \text{Partial}^{\hat{s}_k}_{\Delta(v_1)}(a) + \text{Partial}^{\hat{s}_k}_{\Delta(v_2)}(a) + \hat{s}(h,a),$$

$$\text{Partial}^{\hat{s}_k}_{\mathcal{I}(h)}(a) \leftarrow \text{Partial}^{\hat{s}_k}_{\Delta(v_2)}(a).$$

Finally, since $\mathcal{I}(a) = 0$, $\hat{s}_{k+1}(a,i)$ and $\hat{s}_{k+1}(a,h)$ can be obtained as follows (lines 12–17):

$$\hat{s}_{k+1}(a,x) \leftarrow \frac{C}{2|\mathcal{I}(x)|}\text{Partial}^{\hat{s}_k}_{\mathcal{I}(x)}(a) \qquad (x \in \{i,j\})$$

The rest of the results are shown in Col. 'SR*' in Figure 7.1. □

Exponential SimRank* Optimization. The aforementioned optimization methods for (geometric) SimRank* computation can be readily extended to exponential SimRank*.

To shed light on this, we recall the exponential SimRank* series in Eq.(7.8) and its closed form Eq.(7.12) in Theorem 7.12. Similar to the proof of Theorem 7.12, one can readily show that the k-th partial sum of $\hat{\mathbf{S}}'$ defined by

$$\hat{\mathbf{S}}'_k \triangleq e^{-C} \cdot \sum_{l=0}^{k} \frac{C^l}{l!} \cdot \frac{1}{2^l} \sum_{\alpha=0}^{l} \binom{l}{\alpha} \cdot \mathbf{Q}^{\alpha} \cdot (\mathbf{Q}^T)^{l-\alpha} \tag{7.15}$$

can be represented as the product of the k-th partial sum of matrix exponential $(e^{\frac{C}{2}\mathbf{Q}})$ and its transpose $(e^{\frac{C}{2}\mathbf{Q}})^T$, *i.e.,*

$$\hat{\mathbf{S}}'_k = e^{-C} \cdot \mathbf{T}_k \cdot {\mathbf{T}_k}^T, \text{ with } \mathbf{T}_k \triangleq \textstyle\sum_{i=0}^{k} (\frac{C}{2}\mathbf{Q})^i / i!.$$

Thus, computing $\hat{\mathbf{S}}'_k$ amounts to solving $\mathbf{T}_k$ that can be iteratively derived as follows:

$$\begin{cases} \mathbf{R}_{k+1} = \mathbf{Q} \cdot \mathbf{R}_k \\ \mathbf{T}_{k+1} = \mathbf{T}_k + \frac{C^k}{2^k \cdot k!} \cdot \mathbf{R}_k \end{cases} \text{with} \begin{cases} \mathbf{R}_0 = \mathbf{I}_n \\ \mathbf{T}_0 = \mathbf{0}_n \end{cases}, \tag{7.16}$$

where $\mathbf{R}_k$ is an auxiliary matrix used for computing $\mathbf{T}_k$.

It is worth noting that the matrix equation $\mathbf{R}_{k+1} = \mathbf{Q} \cdot \mathbf{R}_k$ in Eq.(7.16) can be rewritten, in the component form, as

$$\begin{aligned} [\mathbf{R}_{k+1}]_{(a,b)} = [\mathbf{Q} \cdot \mathbf{R}_k]_{(a,b)} &= \sum_{y=1}^{n} [\mathbf{Q}]_{(a,y)} \cdot [\mathbf{R}_k]_{(y,b)} \\ &= \tfrac{1}{|\mathcal{I}(a)|} \sum_{y \in \mathcal{I}(a)} [\mathbf{R}_k]_{(y,b)}, \end{aligned}$$

which takes the similar form of the single summation in Eq.(7.14) except for the coefficient $\frac{C}{2}$. Thus, our previous optimization approach of fine-grained partial sums sharing used for Eq.(7.14) can be applied in a similar way to Eq.(7.16), for improving the computational efficiency. For the interest of space, we omit the detailed algorithm here.

7.4 An Alternative Look of SimRank*

Theorem 7.10 provides a nice interpretation of SimRank*: *Two nodes are similar if their non-punctured in-neighbors are similar.* Here, *non-punctured in-neighbor set* of node a, denoted as $\dot{\mathcal{I}}(a)$, refers to the in-neighbor set of a, plus node a itself, *i.e.*, $\dot{\mathcal{I}}(a) = \mathcal{I}(a) \cup \{a\}$. As opposed to SimRank that is interpreted as "*two nodes are similar if their in-neighbors are similar*", SimRank* infers the similarity of (a, b) not only from similarities of all their in-neighbor pairs $\{(i, j)\}_{i \in \mathcal{I}(a), j \in \mathcal{I}(b)}$, but also from similarities of node-pairs $\{(a, j)\}_{j \in \mathcal{I}(b)}$ and $\{(i, b)\}_{i \in \mathcal{I}(a)}$. Hence, due to SimRank* recursion, the consideration of both $\{(a, j)\}_{j \in \mathcal{I}(b)}$ and $\{(i, b)\}_{i \in \mathcal{I}(a)}$ makes it possible for SimRank* to identify contributions of all *dissymmetric* in-link paths for (a, b).

In accordance with the SimRank philosophy, the interpretation of SimRank* leads itself to have an *informal* SimRank-like representation as follows:

$$\hat{s}(a, b) = \frac{C}{|\dot{\mathcal{I}}(a)||\dot{\mathcal{I}}(b)|} \sum_{j \in \dot{\mathcal{I}}(b)} \sum_{i \in \dot{\mathcal{I}}(a)} \hat{s}(i, j), \tag{7.17}$$

which is the same as SimRank formula Eq.(1.1) except a substitution of $\dot{\mathcal{I}}(\star)$ for $\mathcal{I}(\star)$. However, in component notations, SimRank* in Theorem 7.10 takes a slightly different form:

$$\hat{s}(a,b) = \frac{C}{2\,|\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \hat{s}(a,j) + \frac{C}{2\,|\mathcal{I}(a)|} \sum_{i \in \mathcal{I}(a)} \hat{s}(i,b). \tag{7.18}$$

Comparing the summations in Eqs.(7.17) and (7.18), we observe that they both assess the impact of $\{(a,j)\}_{j \in \mathcal{I}(b)}$ and $\{(i,b)\}_{i \in \mathcal{I}(a)}$ on $\hat{s}(a,b)$, but the double summation in Eq.(7.17) may often suffer from unnecessary "duplicate consideration". To see this, we expand the double summation in Eq.(7.17):

$$\sum_{j \in \dot{\mathcal{I}}(b)} \sum_{i \in \dot{\mathcal{I}}(a)} \hat{s}(i,j) = \overbrace{\sum_{j \in \mathcal{I}(b)} \hat{s}(a,j) + \sum_{i \in \mathcal{I}(a)} \hat{s}(i,b)}^{\text{Part 1}} + \underbrace{\sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} \hat{s}(i,j) + s(a,b)}_{\text{Part 2}}.$$

It can be noted that Part 2 is redundant relative to Part 1. This is because Eq.(7.17) is defined *recursively*, and therefore, the similarities of both $\{(a,j)\}_{j \in \mathcal{I}(b)}$ and $\{(i,b)\}_{i \in \mathcal{I}(a)}$ have recursively "contained" the similarities of $\{(i,j)\}_{i \in \mathcal{I}(a), j \in \mathcal{I}(b)}$ which are duplicates considered in Part 2 again. In contrast, the formalization of SimRank* in Eq.(7.18) has no such redundancy, where the two single summations retain the same efficacy as Part 1 of Eq.(7.17).

7.5 Experimental Evaluation

Our comprehensive empirical studies on real and synthetic data evaluate (i) the semantic richness and relative order of SimRank*; (ii) the computational efficiency of SimRank*.

7.5.1 Experimental Setting

We use the following real and synthetic datasets.

(1) Real data. For semantics and relative order evaluation, we use two graphs: CitHepTh (*directed*) and DBLP (*undirected*).

Dataset	$\|\mathcal{G}\|$ ($\|\mathcal{V}\|, \|\mathcal{E}\|$)	Density ($\|\mathcal{E}\|/\|\mathcal{V}\|$)
CitHepTh	451K (33K, 418K)	12.6
DBLP	102K (15K, 87K)	5.8
D05	21K (4K, 17K)	4.3
D08	85K (13K, 72K)	5.5
D11	103K (14K, 89K)	6.3
Web-Google	5.8M (873K, 4.9M)	5.6
CitPatent	19.8M (3.6M, 16.2M)	4.5

Figure 7.5: Details of Real Datasets

(a) CitHepTh[12], a citation network, where nodes are papers labeled with titles, and an edge a citation. The data is collected from the arXiv, with papers from 1993 to 2003.

(b) DBLP[13], a collaboration graph, where nodes are authors, and edges co-authorships. The graph is derived from 6-year publications (2002–2007) in seven major conferences: SIGMOD, PODS, VLDB, ICDE, SIGKDD, SIGIR, WWW.

For computational efficiency evaluation, we use five graphs:

(c) D05, D08, D11, three co-authorship graphs, which are constructed from 9-year DBLP publications (2003–2011) in 7 major conferences (as remarked in the first DBLP dataset). Each graph is built by choosing every 3 years as a time step.

(d) Web-Google, a web graph, where nodes are pages, and edges links. The data is from Google Programming Contest.

(e) CitPatent, a U.S. patent network, in which nodes are patents, and edges are citations made by patents. This data is maintained by the National Bureau of Economic Research.

The size $|G|(|\mathcal{V}|, |\mathcal{E}|)$ of the graphs are shown in Figure 7.5.

(2) Synthetic data. To produce synthetic networks, we use a generator GTgraph[14] that is controlled by $|\mathcal{V}|$ and $|\mathcal{E}|$.

(3) Baselines. We implement the following algorithms in Visual C++ 9.0. (a) our geometric SimRank* algorithm memo-gSR* and it exponential variant memo-eSR* via

[12]http://snap.stanford.edu/data/index.html
[13]http://dblp.uni-trier.de/~ley/db/
[14]http://www.cse.psu.edu/~madduri/software/GTgraph/index.html

fine-grained memoization (Section 7.3.3); (b) our conventional iterative SimRank* algorithm iter-gSR* which, as a comparison to memo-gSR*, computes similarities without memoization (Section 7.3.2); (c) psum-SR [LVGT08] and psum-PR [ZHS09] algorithms that compute SimRank and P-Rank similarities via partial sums memoization, respectively; (d) mtx-SR algorithm [LHH+10] that computes SimRank using singular value decomposition. (e) RWR [TFP06] measures the node proximity *w.r.t.* a query.

(4) Test Queries. To serve the ranking purpose, we select 500 query nodes from each graph, based on the following: For each graph, we first sort all nodes in order of their in-degree into 5 groups, and then randomly choose 100 nodes from each group, aiming to guarantee that the selected nodes can systematically cover a broad range of all possible queries. Here, we mainly focus on single-node queries, since a multi-node query can be fairly factorized into multiple single-node queries via Linearity Theorem [Cha07]. For every experiment, the average performance is reported over all test queries.

(5) Parameters. We set the following default parameters: (a) $C = 0.6$, which is the typical decay factor used in [JW02]. (b) $K = 5$, which is the total number of iterations, being the time-accuracy trade-off. Besides, for all the methods, we clip similarity values at 10^{-4}, to discard far-apart nodes with scores less than 10^{-4} for storage. It can greatly reduce space cost with minimal impact on accuracy, as shown in [LVGT08].

(6) Effectiveness Metrics. To evaluate semantics and relative ordering, we consider both node and node-pair ranking. We adopt three metrics [LHH+10, Cha07]: *Kendall's* τ, *Spearman's* ρ, and *Normalized Discounted Cumulative Gain* (NDCG).

(a) *Kendall's* τ is defined as

$$\tau = \frac{2}{N(N-1)} \sum_{\{i,j\} \in P} \bar{K}_{i,j}(\tau_1, \tau_2),$$

with $\bar{K}_{i,j}(\tau_1, \tau_2) = 1$ if i and j are in the same order in τ_1 and τ_2, and otherwise 0. Here, τ_1 and τ_2 are the rankings of elements in two lists, P is the set of unordered pairs in τ_1 and τ_2, and N is the number of elements in a ranking list.

(b) *Spearman's* ρ is given by

$$\rho = 1 - \frac{6 \sum_{i=1}^{N} d_i^2}{N(N^2-1)},$$

where d_i is the ranking difference between the i-th elements in two lists.

(c) *NDCG at position p w.r.t. query q* is given by

$$\text{NDCG}_p(q) = \frac{1}{\text{IDCG}_p(q)} \sum_{i=1}^{p} \frac{2^{s(i,q)} - 1}{\log_2(1+i)},$$

where $s(i,q)$ is the similarity score between nodes i and q, and $\text{IDCG}_p(q)$ is a normalized factor ensuring the "true" NDCG ordering to be 1.

(7) Ground Truth. (a) To validate similar authors on DBLP, we invite 20+ experts from database and data mining areas to assess the "true" relevance of each retrieved co-authorship. They may also refer to Co-Author Path in Microsoft Academic Search[15] to see "separations" between collaborators. (b) To evaluate similar papers on CitHepTh, we hire 15+ researchers from the physical department for judging the "true" relevance of the retrieved co-citations. Their assessment may hinge on paper contents, H-index, and #-citations in `www.ScienceDirect.com`. For all the ground truths, the final results are rendered by a majority vote of feedbacks.

All experiments are run on a machine powered by an Intel Core(TM) 3.10GHz CPU with 8GB RAM, on Windows 7.

7.5.2 Experimental Results

We next present our empirical findings.

Exp-1: Semantics & Relative Order.

We first run the algorithms on *directed* CitHepTh and *undirected* DBLP. By randomly issuing 500+ queries, we evaluate the average semantic accuracy for each algorithm via three metrics (Kendall, Spearman, NDCG). Figure 7.6 depicts the results. (Due to space limits, many case studies are reported in [YLZ+13b] to further exemplify the quantitative

[15] http://academic.research.microsoft.com/VisualExplorer

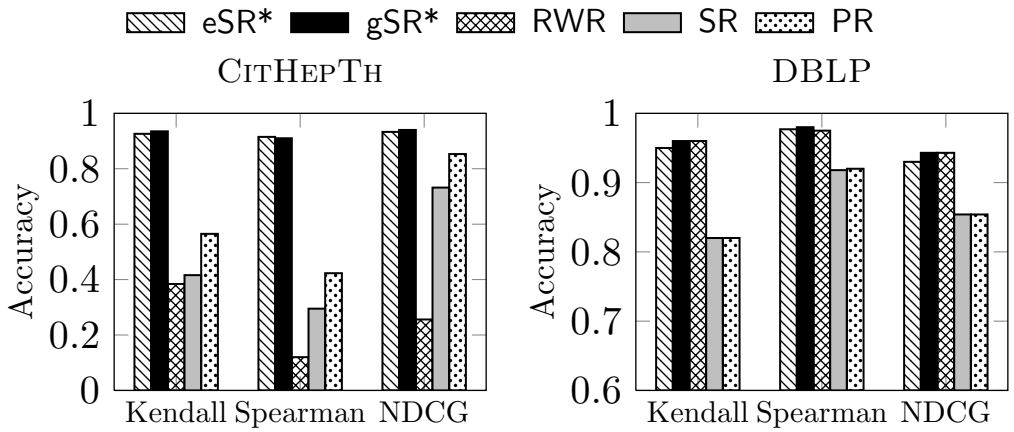


Figure 7.6: Semantic Effectiveness on Real Data

Q1: Prof. Jennifer Widom

#	SR* / RWR	SR
1	**Chris Olston (46)**	David Konopnicki
2	**Glen Jeh (47)**	Ying Xu
3	**Oded Shmueli (55)**	Nina Mishra
4	David Konopnicki	K. Kenthapadi
5	**Pedro Bizarro (19)**	Gagan Aggarwal
6	Ying Xu	Dilys Thomas

Q2: Prof. Alon Y. Halevy

#	SimRank* / RWR	SR
1	**Jun Zhang (11)**	Steven D. Gribble
2	**Ema Nemes (10)**	Oren Etzioni
3	**Xin Dong (13)**	Luke McDowell
4	Deepak Verma	Stani Vlasseva
5	Stani Vlasseva	Deepak Verma
6	William Pentney	William Pentney

Q3: Dr. Rakesh Agrawal

#	SR* / RWR	SR
1	Adina Crainiceanu	Yirong Xu
2	**Byron Dom (13)**	Roberto J. Bayardo
3	Vuk Ercegovac	Vuk Ercegovac
4	Kristen LeFevre	Kristen LeFevre
	...	...
12	**Christos Faloutsos (36)**	Ralf Rantzau
13	**S. Chakrabarti (53)**	Byron Dom
14	**Mayank Bawa (56)**	Sridhar Rajagopalan
15	**R. Ramakrishnan (45)**	A. V. Evfimievski

Q4: Top-4 Similar Author-Pairs (SR*/RWR)

1	Israel Cidon	Oren Unger
2	Yuval Shavitt	Xiaohua Jia
3	Daniel A. Menasc	Harry J. Foxwell
4	Bo Li	Jiangchuan Liu

Q4: Top-4 Similar Author-Pairs (SR)

1	Milena Ivanova	Gustav Fahl
2	Wanhong Xu	Naci Ishakbeyoglu
3	Shengli Sheng	Jin Huang
4	Kar Wing Li	Fu Lee Wang

Figure 7.7: Case Study: Top Co-authors on DBLP (2003–2005)

results in Figure 7.6.) (1) On CitHepTh, memo-gSR* and memo-eSR* have higher accuracy (*e.g.,* Spearman's $\rho \approx 0.91$) than psum-SR (0.29), RWR (0.12) and psum-PR (0.42) on average, *i.e.,* the semantics of SimRank* is effective. This is because SimRank* considers *all* in-link paths for assessing similarity, whereas SimRank and RWR, respectively, counts only limited *symmetric* and *unidirectional* paths. (2) On DBLP, the accuracy of RWR is the same as memo-gSR* and memo-eSR*, due to the *undirectedness* of DBLP. This tells us that, regardless of edge directions, both SimRank* and RWR count the

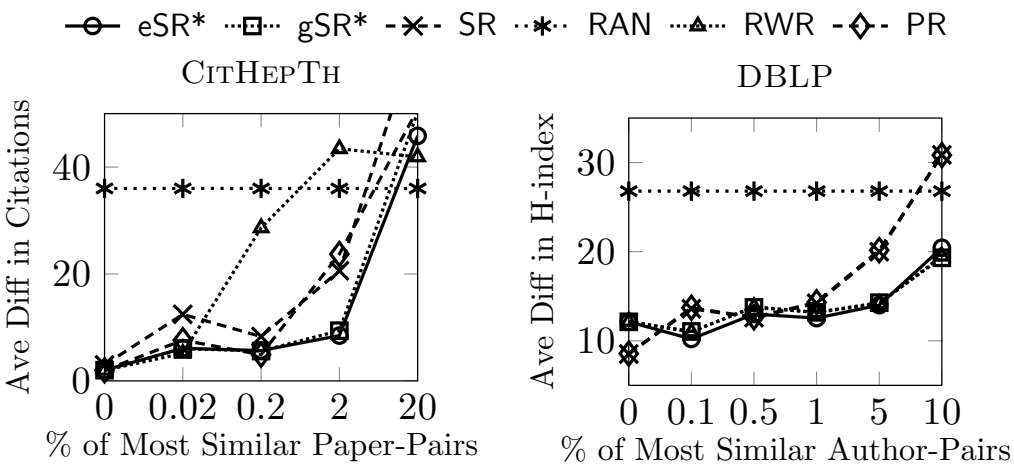


Figure 7.8: Role Difference of Top Ranked Node-Pairs

path of *all* lengths, as opposed to SimRank considering only the *even*-length paths. Likewise, psum-PR and psum-SR produce the same results on undirected DBLP. (3) On both datasets, memo-gSR* and memo-eSR* keep almost the same accuracy, implying that the relative order of the geometric SimRank* is well maintained by its exponential counterpart.

Figure 7.7 reveals four top-k query results on DBLP. For example, Q1 finds most similar co-authors of Prof. Jennifer Widom via memo-gSR*, memo-eSR*, RWR , psum-SR. We observe the following. (1) RWR and memo-gSR* have the same results on DBLP, which is due to the undirectedness of DBLP, as expected. (2)memo-gSR* and memo-eSR* also yield the results, showing the relative ranking preservation of memo-eSR* *w.r.t.* memo-gSR* for top-k results. (3) Many closely related co-authors of Widom with an undesirably low rank via psum-SR (as shown in brackets of gray cells) can be well identified by memo-gSR*, memo-eSR*, RWR . This is because SimRank ignores contributions of *dissymmetric* in-link paths (*i.e.,* paths of odd lengths in undirected graphs), whereas SimRank* considers contributions of *all* in-link paths. The disparity of ranking in gray cells shows that memo-gSR*, memo-eSR*, RWR can perfectly resolve the "zero-similarity" issue of psum-SR on undirected graphs.

Figure 7.8 further validates that node-pairs with *high* SimRank* scores do have *similar* roles. On CitHepTh, we use #-citation as a proximity measure for co-citation role; on

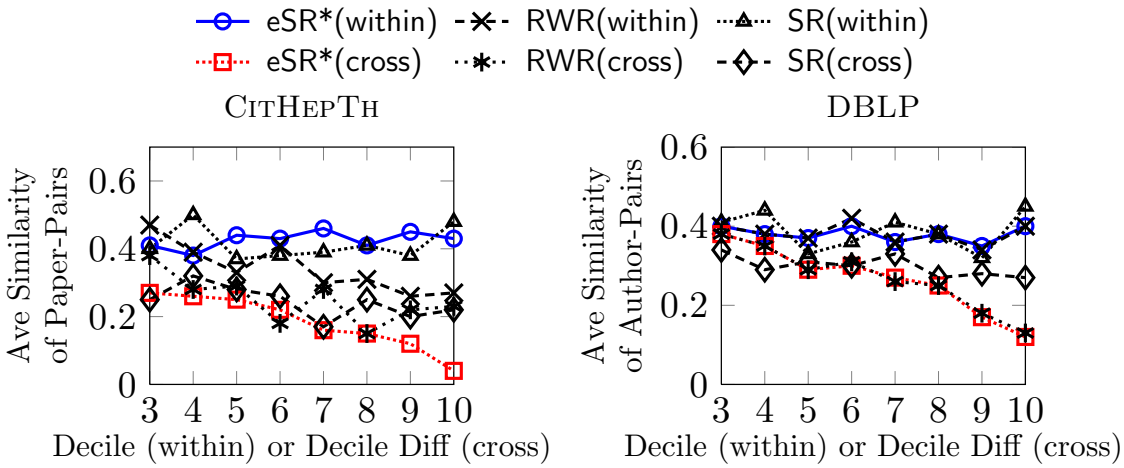


Figure 7.9: Average Similarity of Grouped Node-Pairs

DBLP, we use H-index for coauthor role, since if a paper is highly cited, it will increase the H-index of every co-author. From the results, we see that on CitHepTh, for the top 2% similar paper-pairs, the average difference in their #-citation is 8 for memo-gSR* and memo-eSR*, which is lower than psum-SR (21), psum-PR (24), RWR (43), and the random-pair difference RAN (38). A *lower* average difference in #-citation (*resp.*H-index) indicates that papers (*resp.*authors) are reliably *similar*. As we increase the search to top 20% similar paper-pairs on CitHepTh, SimRank* can constantly find *reliable* similarity, whereas SimRank converges to random scoring. Thus, node-pairs with higher SimRank* scores will have similar roles. A similar result is shown on DBLP.

Figure 7.9 confirms that nodes with similar roles do have high SimRank* scores. On CitHepTh (*resp.*DBLP), we group the papers (*resp.*authors) into 10 roles based on the #-citation (*resp.*H-index), from top 10% to bottom 10%. For each node-pair, if two nodes are within the same role, we average out their similarity score for this role. We also average out #-node-pairs not within the same role (across roles). We see that *e.g.,* on DBLP, the average SimRank* similarity within the same role is stable around 0.4, in contrast with SimRank fluctuating between 0.35 and 0.45, due to many dissymmetric paths completely neglected by SimRank. For the author-pairs across roles, the x-axis denotes the difference of role decile for two authors in a pair. The decreasing line of memo-eSR* and RWR indicates that role similarity correctly decreases as H-index gets

Q1: "The Relativistic Dirac-Morse Problem via SUSY QM" #-Res	19	0	3
Paper Title	SR*	RWR	SR
Comment on Solution of the Relativistic Dirac-Morse Problem	1	-	-
Confinement of fermions by mixed vector-scalar linear potentials in two ...	2	-	1
Comment on "Relativistic extension of shape-invariant potentials"	3	-	2
Bound states by a pseudoscalar Coulomb potential in 1+1 dims	4	-	-
The Quantum Mechanics SUSY Algebra: An Introductory Review	5	-	3
Nonlinear supersymmetry in Quantum Mechanics: algebraic ...	6	-	-
(Super)Oscillator on CP(N) and Constant Magnetic Field	7	-	-

Q2: "Time Evolution via S-branes" #-Res	3869	16	2497
Paper Title	SR*	RWR	SR
Tachyon Matter	1	-	**91**
Cosmological Evolution of the Rolling Tachyon	2	-	4
Non-Standard Intersections of S-Branes in D=11...	3	1	**19**
Inflation from a Tachyon Fluid?	4	-	2
Causality and the Skyrme Model	5	-	1
On the Deconstruction of Time	6	-	**101**
A Torsion Correction to RR 4-Form Fieldstrength	7	-	**100**

Q3: Top-2 Most Similar Paper-Pairs (SR*, RWR, SR)

Classification and Quantum Moduli Space of D-branes in Group Manifolds	A note on singular D-branes in group manifolds	1	SR*
New representation for Lagrangians of self-dual nonlinear electrodynamics	Helicity conservation in Born-Infeld theory	2	
Classification and Quantum Moduli Space of D-branes in Group Manifolds	A note on singular D-branes in group manifolds	1	RWR
Towards higher-N superextensions of Born-Infeld theory	New representation for Lagrangians of self-dual nonlinear electrodynamics	2	
Romans type IIA theory and the heterotic strings	Charging Interacting Rotating Black Holes in Heterotic String Theory	1	SR
Towards higher-N superextensions of Born-Infeld theory	Invariance of interaction terms in new representation of self-dua electrodynamics	2	

Figure 7.10: Case Study: Top Co-Citations on CitHepTh

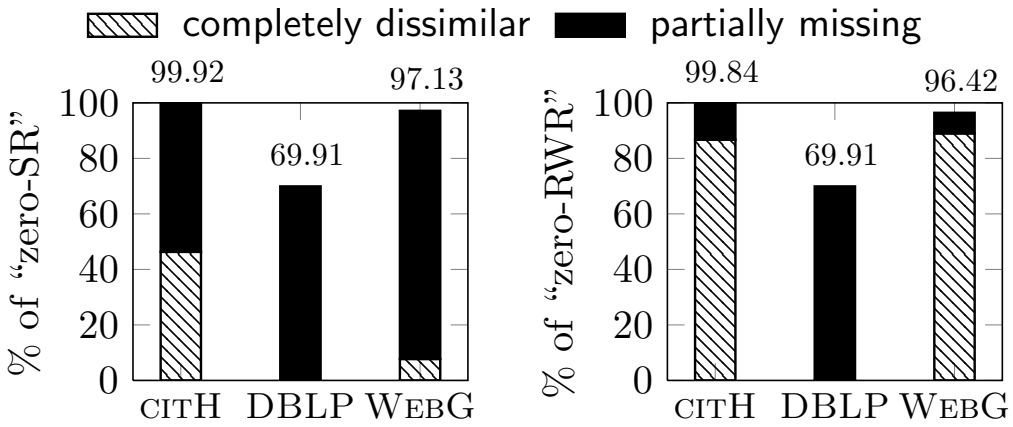


Figure 7.11: % of "Zero-Similarity" Node-Pairs on Real Data

less similar. For psum-SR, the average across-role similarity is round 0.3, approaching random scoring. This tells that SimRank* scores are more reliable than others to reflect nodes with similar roles. The result is more pronounced on CitHepTh.

To further compare the difference between RWR and memo-gSR*, we use the *directed* graph CitHepTh, and issue four paper queries. The search results are shown in Figure 7.10. It should be noted that for directed CitHepTh, RWR and memo-gSR* have substantial differences: For the 1st query, RWR fails to find any results, whereas memo-gSR* perfectly finds useful papers, better than SimRank. This is because RWR only considers unidirectional paths between two nodes, thus limiting its utility for find sensible papers, whereas SimRank* considers all in-link paths. Other results on SimRank* and SimRank are analogous to those on DBLP. Due to space limits, we omit it here.

Figure 7.11 shows the "zero-similarity" issues for SimRank and RWR *commonly* exist in real graphs. The results on *e.g.,* CitHepTh show that more than 95% of node-pairs have "zero-SimRank" issues, among which about 40% are assessed as "completely dissimilar" (*i.e.,* SimRank=0), and about 55% have "partially missing" issue (SimRank $\neq$0, but miss the contributions of the dissymmetric in-links paths). It shows the necessity for our revision of SimRank and RWR.

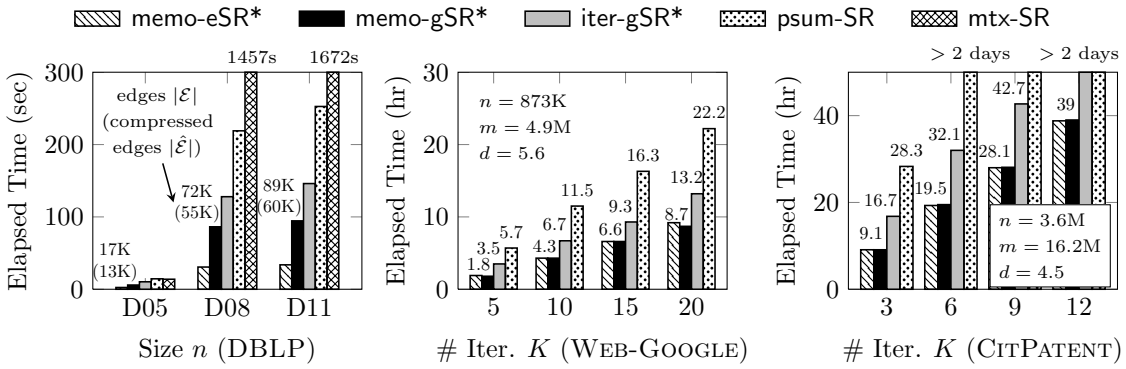


Figure 7.12: Time Efficiency on Real Datasets

Exp-2: Time Efficiency.

We next evaluate (1) the CPU time of SimRank* on real data, and (2) the impact of graph density on CPU time on synthetic data.

Fixing accuracy $\epsilon = .001$ on DBLP, and varying K on Web-Google and CitPatent, we compare the CPU time of the five algorithms. The results are shown in Figure 7.12, telling the following. (1) In all the cases, memo-gSR* and memo-eSR* outperform iter-gSR*, psum-SR and mtx-SR, *i.e.,* our fine-grained memoization approach is efficient. Indeed, mtx-SR is the slowest on D05, D08, D11 due to its cost-inhibitive SVD. On Web-Google, memo-gSR* (memo-eSR*) is on average 1.6X and 2.6X faster than iter-gSR* and psum-SR, respectively. On CitPatent, the speedup of memo-gSR* (memo-eSR*) is on average 1.7X and 3.1X better than iter-gSR* and psum-SR, respectively. When $K \geq 6$, psum-SR takes too long to finish computations in two days on large CitPatent, which is practically unacceptable. In contrast, memo-gSR* (memo-eSR*) just needs about 19.5 hours for $K = 6$. This is because SimRank* takes a simpler form than SimRank, in which one just needs to compute one *single* summation per iteration, in contrast to a *double* summation of psum-SR. (2) Given $\epsilon = .001$ on DBLP, the speedup of memo-eSR* is more pronounced, 6.8X, 4.2X, 2.7X faster than psum-SR, iter-gSR*, memo-gSR* on average, respectively. This is because the closed matrix form of memo-eSR* accelerates the convergence of SimRank*, thus yielding less iterations for attaining the same accuracy ϵ.

Figure 7.13 further shows the amortized time for each phase of memo-eSR* and memo-

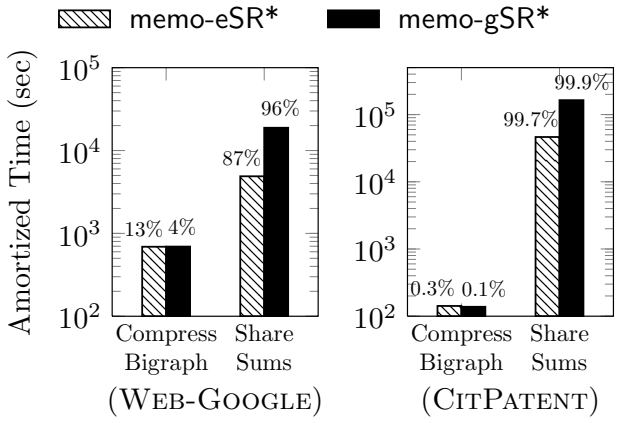


Figure 7.13: Amortized Time on Real Data

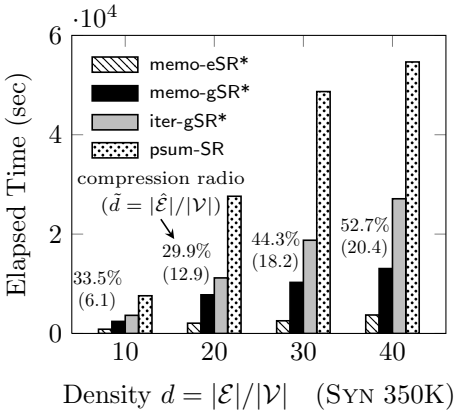


Figure 7.14: Effect of Density

gSR* on Web-Google and CitPatent (given $\epsilon = .001$), with x-axis being two phases. From the results, (1) for memo-eSR* and memo-gSR*, the time for "Compress Bigraph" is about one order of magnitude less than the time for "Share Sums" on Web-Google, and 2.5 orders of magnitude less on CitPatent. This tells that the preprocessing does not incur much extra time, confirming our complexity analysis in Subsect. 7.3.3. (2) "Compress Bigraph" takes up larger portions (13% on Web-Google, and 0.3% on CitPatent) in the total time of memo-eSR*, than those (4% on Web-Google, and 0.1% on CitPatent) in the total time of memo-gSR*. This is because memo-eSR* and memo-gSR* takes (almost) the same time for "Compress Bigraph", whereas, for "Share Sums", memo-eSR* needs less time (3.8X on Web-Google, 3.5X on CitPatent) than memo-gSR*, due to the convergence speedup of memo-eSR*.

Fixing $n = 350$K, varying m from 3.5M to 14M on synthetic data, Figure 7.14 shows the impact of graph density $d = m/n$ on CPU time. The results show that (1) given $\epsilon = .001$, memo-eSR* outperforms memo-gSR*, iter-gSR*, and psum-SR by 3.5X, 6.1X, and 14X speedups, respectively, as m increases. (2) The speedups of memo-eSR* and memo-gSR* are sensitive to graph density. This is because when graphs become denser, there is a higher likelihood that in-neighbor sets will overlap one another for fine-grained partial sums sharing. The biggest speedups are observed for higher density — with nearly 1.5 orders of magnitude speedup at $d = 40$, and its compression ratio is 52.7%. [16]

[16]Here, the compression ratio is defined by $(1 - \frac{\tilde{m}}{m}) \times 100\%$, where $\tilde{m}$ is the number of edges in the

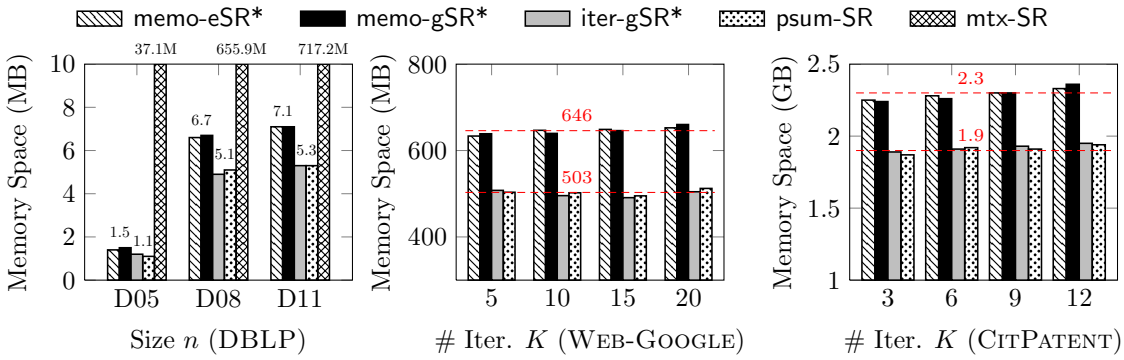


Figure 7.15: Memory Space on Real Datasets

Exp-3: Memory Space.

Lastly, we evaluate the space requirement of memo-eSR* and memo-gSR* against iter-gSR*, psum-SR and mtx-SR on real data. We only use mtx-SR on small DBLP since its memory space will explode on large Web-Google, due to its SVD destroying graph sparsity.

The results are reported in Figure 7.15. We observe that (1) in all the cases, memo-eSR* and memo-gSR* take almost the same space, both of which fairly retain the same orders of magnitude as iter-gSR* and psum-SR. Indeed, both memo-eSR* and memo-gSR* only need 28.2%, 29.3%, and 19.2% more space on average on DBLP, Web-Google, and CitPatent, respectively, as compared with iter-gSR* and psum-SR. The extra space of memo-eSR* and memo-gSR* is used for fine-grained SimRank* memoization. This tells that memo-eSR* and memo-gSR* do not need to sacrifice much space for achieving high time efficiency. (2) On DBLP, memo-eSR* and memo-gSR* require far less space than mtx-SR by at least one order of magnitude, since mtx-SR using SVD may produce very dense matrices. (3) On Web-Google and CitPatent, the space of memo-eSR* and memo-gSR* is stable as K grows because the memoized partial sums are immediately released after each iteration.

compressed graph $\hat{\mathcal{G}}$.

7.6 Related Work

We categorize related work as follows.

Link-based Similarity. One of the most renowned link-based similarity metrics is SimRank, invented by Jeh and Widom [JW02]. It iteratively captures the notion that "two nodes are similar if they have *similar* in-neighbors", which weakens the philosophy of the rudimentary measures (*e.g.,* Coupling [Kes63a], Co-citation [Sma73]) that "two nodes are similar if they have the *same* neighbors in common". The recursive nature of SimRank allows *two* nodes to be similar without common in-neighbors, which resembles PageRank [Ber05] assigning a relevance score for *each* node. SimRank implies an unsatisfactory trait: The similarity of two nodes decreases as the number of their common in-neighbors increases. To address this issue, Fogaras and Rácz [FR05] introduce P-SimRank. They (1) incorporate Jaccard coefficients, and (2) interpret $s(a, b)$ as the probability that two random surfers, starting from a and b, will meet at a node. Antonellis *et al.* [AMC08] propose SimRank++, by adding an evidence weight to compensate for the cardinality of in-neighbor matching. MatchSim [LLK12] refines SimRank with maximum neighborhood matching. RoleSim [JLH11] deploys generalized Jaccard coefficients to ensure automorphic equivalence for SimRank. However, none of them resolves the "zero-SimRank" issue. This issue surfaces in part in the motivating Example 1.2 of Zhao *et al.* [ZHS09] who propose P-Rank taking both in- and out-links into account. Our work differs from [ZHS09] in that (1) we show that the "zero-SimRank" issue is not caused by the ignorance of out-links in SimRank, and (2) we circumvent the "zero-similarity" issue by traversing more incoming paths of node-pairs that are neglected by the original SimRank.

There has also been work on link-based similarity (*e.g.,* [YHY06, BGH+04, XFF+05, TFP06, LHN06]). LinkClus [YHY06] uses a hierarchical structure, called SimTree, for clustering multi-type objects. Blondel *et al.* [BGH+04] propose an appealing measure to quantify *graph* similarities. SimFusion [XFF+05] utilizes a reinforcement assumption for assessing similarities of multi-type objects in a heterogenous domain, as opposed to SimRank focusing solely on intra-type objects in a homogenous domain. Tong *et al.* [TFP06]

suggest Random Walk with Restart (RWR) for assessing node proximities, which is an excellent extension of Personalized PageRank (PPR). Leicht *et al.* [LHN06] extend RWR by incorporating independent and sensible coefficients. However, RWR and its variants (PPR and [LHN06]) also imply SimRank-like "zero-similarity" issues, as discussed in Subsect. 7.2.1.

Similarity Computation. The computational overheads of link-based similarity often arise from its recursive nature. To meet this challenge, Lizorkin *et al.* [LVGT08] propose three excellent optimization methods to SimRank (*i.e.,* essential node-pair selection, partial sums memoization, and threshold-sieved similarities). These substantially speed up SimRank computation from $O(Kd^2n^2)$ to $O(Knm)$ time, with d being the average in-degree of a graph. In contrast, our model performs even faster than SimRank, yet can enumerate more incoming paths missed by SimRank to enrich semantics since (1) our model can be simplified into a much simpler form than SimRank, and (2) the computation can be further accelerated via fine-grained memoization. Li *et al.* [LHH+10] use graph low-rank structure to compute SimRank via singular value decomposition (SVD), yielding $O(r^4n^2)$ time, with r ($\leq n$) being the rank of an adjacency matrix. However, it does not always reduce the complexity when r is large. In contrast, SimRank* needs $O(Kn\tilde{m})$ worst-case time, with $\tilde{m} \leq m$. He *et al.* [HLC+12] study the incremental SimRank with the focus on node updates for parallel computing on GPU.

7.7 Conclusions

In this chapter, we have proposed SimRank*, a refinement of SimRank, for effectively assessing link-based similarities. In contrast to SimRank only considering contributions of *symmetric* in-link paths, SimRank* can tally contributions of *all* in-link paths between two nodes, thus resolving the "zero-SimRank" issue for semantic richness. We have also converted the series form of SimRank* into two elegant forms: the geometric SimRank* and its exponential variant, both of which look even simpler than SimRank, yet without suffering from increased computational cost. Finally, we have developed

a fine-grained memoization strategy via edge concentration, with an efficient algorithm speeding up SimRank* computation from $O(Knm)$ to $O(Kn\tilde{m})$ time, where $\tilde{m}$ is generally much smaller than m. Our experimental results on real and synthetic data show richer semantics and higher computation efficiency of SimRank*.

Chapter 8

Conclusions and Future Work

This thesis studied several efficient techniques for assessing link-based node-to-node relevance on large and dynamic networks, including fast SimRank, SimFusion and P-Rank assessment on large networks, incremental SimRank, RWR and SimFusion+ update on dynamic networks, and the development of novel relevance scoring mechanisms for enriching semantics. In the following, we summarize the main contributions of this thesis in Section 8.1, and discuss general avenues for future research arising from this thesis in Section 8.2.

8.1 Thesis Summary

Chapter 1 highlighted the need for efficiently assessing node-to-node relevance through motivating applications, and presented necessary fundamentals of relevance assessment. The main technical contributions of the thesis appeared in Chapters 2–7.

- **Fast SimRank Assessment on Large Networks and Bipartite Domains.** Chapter 2 proposed efficient methods for speeding up SimRank assessment on large networks, as well as bipartite domains. For large networks, (1) a fine-grained partial sums sharing approach was presented to improve the SimRank computation per iteration; (2) To reduce the total number of iterations, a new differential SimRank equation was proposed, which can represent the SimRank solution as an expo-

nential sum of the transition matrix, rather than the geometric counterpart. For bipartite domains, (3) the techniques for optimizing Minimax SimRank variation computation was also proposed by utilizing edge concentration technique.

- **Incremental SimRank on Link-Evolving Networks.** Chapter 3 presented an innovative paradigm that supports incremental updates for SimRank assessment over dynamic networks, where links frequently evolve over time. (1) The SimRank update matrix, in response to each link update, was first represented as a rank-one Sylvester equation. (2) By virtue of this, an effective pruning technique was also leveraged for capturing the "affected areas" of the SimRank update matrix to skip unnecessary computations, without loss of exactness.

- **P-Rank Assessment on Large Networks.** Chapter 4 provided fast approaches for accelerating P-Rank assessment on large networks. (1) The accuracy estimate was given for P-Rank iterations, aiming to find out the total number of iterations needed for a given accuracy. (2) The stability of P-Rank was also analyzed to find out the condition under which P-Rank is stable. (3) Two efficient algorithms were devised for speeding up P-Rank assessment over directed and undirected networks.

- **Incremental RWR on Dynamic Networks.** Chapter 5 studied the incremental RWR update on link-evolving networks. (1) For unit link update, a fast and exact paradigm was devised, which characterizes the RWR vector with respect to a given query in terms of a linear combination of other old RWR vectors. (2) For batch link updates, an efficient algorithm was proposed for incrementally assessing RWR proximities in response to a mix of edge insertions and deletions, without loss of exactness.

- **SimFusion+ on Large and Dynamic Networks.** Chapter 6 addressed two problems (*i.e.,* trivial solution and divergence issue) to the existing SimFusion model, and proposed a full treatment for SimFusion. (1) A fast and exact algorithm was presented to significantly improve the time and space complexity of SimFusion+ assessment on large networks. (2) An approximate algorithm was also

provided for further speeding up SimFusion+ assessment with accuracy guarantees. (3) An incremental eigen-based algorithm was devised to support incremental SimFusion+ updates on dynamic graphs.

- **A Novel Relevance Model for Semantic Richness.** Chapter 7 proposed a novel relevance SimRank* for enriching semantics of SimRank and RWR while inheriting merits of their original basic philosophies. (1) An undesirable "zero-similarity" property in SimRank and RWR was observed, and a novel SimRank* model was proposed for fixing this issue. (2) An elegant and succinct closed-form of SimRank* was introduced for efficient SimRank* assessment on large graphs. (3) A heuristic approach was also presented for further accelerating SimRank* assessment via network compression, without any loss of exactness.

8.2 Future Avenues

Aside from the specific open problems mentioned in individual chapters, there are some general directions to extend the work presented in this thesis:

- **Parallel Relevance Assessment on Distributed Systems.** The entire thesis only dealt with efficient relevance assessment on a single computer. While many optimization techniques described in Chapters 2–7 can be implemented by a single CPU, distributed systems are more appropriate for some concurrent computing applications. For instance, the high memory bandwidth of graphics processing units can be fully utilized to further speed up relevance assessment on large networks. Moreover, parallel relevance computing paradigms can be readily generalized to node-updating algorithms for incremental networks, where nodes constantly arrive over time. The extension of all our results to parallel relevance computing is perhaps the most overarching avenue of future work.

- **Improving I/O costs for Disk-Resident Networks.** For large disk-resident networks, the main overhead for an external relevance algorithm to assess relevance

might be the expensive I/O costs. Hence, another possible extension for the thesis is to develop novel fast relevance computing paradigms that support large-scale disk-resident graphs. Since such graphs could not be held into the main memory, frequent random access from disk demand novel scalable algorithms to reduce I/O costs for relevance assessment.

- **Relevance Assessment on Hypergraphs and Signed Networks.** Networks considered in this thesis are assumed to be graphs whose edges are pairs of nodes, and corresponding adjacency matrices are non-negative, representing a binary 0-1 relationship. There are two directions to extend this work: (1) One possible interesting direction is to extend our results for richer graph models for relevance assessment, *e.g.,* hypergraphs whose (hyper) edges are arbitrary sets of nodes. (2) Another promising avenue is to carry our proposed methodology forward to the setting of social networks with positive and negative links. Such an extension is non-trivial since many theories we studied in previous chapters rely on an underlying assumption that all entries in adjacency matrices are non-negative, *e.g.,* Proposition 6.5 in Chapter 6. This assumption is quite reasonable for standard digraphs, yet does not hold for signed networks, which can be integrated as an interesting direction of further work.

- **Top-k Search for Ranking Purpose.** Throughout the thesis, all our methods are developed for all-pairs relevance assessment. However, in some real applications, to serve the ranking purpose, users are more interested in top-k search results rather than the entire ranking list. Therefore, top-k search with respect to a given query is likely to play a key role in ranking objects on large networks. Extending our proposed methods to top-k search is an overarching research goal for future study.

Bibliography

[AMC08] Ioannis Antonellis, Hector Garcia Molina and Chichao Chang. SimRank++: Query rewriting through link analysis of the click graph. *PVLDB*, 1(1), 2008.

[Ams72] Robert Amsler. Applications of citation-based automatic classification. *Linguistics Research Center*, 1972.

[AP98] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Society for Industrial and Applied Mathematics, 1998.

[BC08a] R.A. Brualdi and D. Cvetkovic. *A Combinatorial Approach to Matrix Theory and Its Applications*. Discrete Mathematics and Its Applications. Taylor & Francis, 2008.

[BC08b] Gregory Buehrer and Kumar Chellapilla. A scalable pattern mining approach to web graph compression with communities. In *WSDM*, 2008.

[BCG10] Bahman Bahmani, Abdur Chowdhury and Ashish Goel. Fast Incremental and Personalized PageRank. *PVLDB*, 4(3), 2010.

[BCX11] Bahman Bahmani, Kaushik Chakrabarti and Dong Xin. Fast personalized PageRank on MapReduce. In *SIGMOD Conference*, pages 973–984, 2011.

[Ber05] Pavel Berkhin. Survey: A Survey on PageRank Computing. *Internet Mathematics*, 2(1), 2005.

[BGH+04] Vincent D. Blondel, Anahí Gajardo, Maureen Heymans, Pierre Senellart and Paul Van Dooren. A Measure of Similarity between Graph Vertices: Applications to Synonym Extraction and Web Searching. *SIAM Rev.*, 46(4), 2004.

[Cha07] Soumen Chakrabarti. Dynamic personalized PageRank in entity-relation graphs. In *WWW*, pages 571–580, 2007.

[Col74] H. M. Collins. The TEA set: Tacit knowledge and scientific networks. *Science Studies*, 4:165–186, 1974.

[CR04] Junghoo Cho and Sourashis Roy. Impact of search engines on page popularity. In *WWW*, 2004.

[CZDC10] Yuanzhe Cai, Miao Zhang, Chris H. Q. Ding and Sharma Chakravarthy. Closed form solution of similarity algorithms. In *SIGIR*, pages 709–710, 2010.

[CZHY12] Hong Cheng, Yang Zhou, Xin Huang and Jeffrey Xu Yu. Clustering large attributed information networks: An efficient incremental computing approach. *Data Min. Knowl. Discov.*, 25(3):450–477, 2012.

[DPSK05] Prasanna Kumar Desikan, Nishith Pathak, Jaideep Srivastava and Vipin Kumar. Incremental PageRank computation on evolving graphs. In *WWW*, 2005.

[Eom96] S. B. Eom. Mapping the intellectual structure of research in decision support systems through author cocitation analysis. *Decision Support Systems*, 16(4):315–338, 1996.

[FNOK12] Yasuhiro Fujiwara, Makoto Nakatsuji, Makoto Onizuka and Masaru Kitsuregawa. Fast and Exact Top-k Search for Random Walk with Restart. *PVLDB*, 5:442–453, 2012.

[FNS+13] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, Takeshi Mishima and Makoto Onizuka. Efficient ad-hoc search for personalized PageRank. In *SIGMOD Conference*, pages 445–456, 2013.

[FNSO13] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa and Makoto Onizuka. Efficient search algorithm for SimRank. In *ICDE*, pages 589–600, 2013.

[FR04] Dániel Fogaras and Balázs Rácz. A Scalable Randomized Method to Compute Link-Based Similarity Rank on the Web Graph. In *EDBT Workshops*, 2004.

[FR05] Dániel Fogaras and Balázs Rácz. Scaling link-based similarity search. In *WWW*, 2005.

[FR07] Dániel Fogaras and Balázs Rácz. Practical Algorithms and Lower Bounds for Similarity Search in Massive Graphs. *IEEE Trans. Knowl. Data Eng.*, 19:585–598, 2007.

[GB09] Bela Gipp and J. Beel. Citation Proximity Analysis (CPA) - A New Approach for Identifying Related Work Based on Co-Citation Analysis. In *ISSI*, 2009.

[GGCM09] Sanchit Garg, Trinabh Gupta, Niklas Carlsson and Anirban Mahanti. Evolution of an online social aggregation network: An empirical study. In *Internet Measurement Conference*, 2009.

[GGST86] Harold N. Gabow, Zvi Galil, Thomas H. Spencer and Robert Endre Tarjan. Efficient algorithms for finding minimum spanning trees in undirected and directed graphs. *Combinatorica*, 6:109–122, 1986.

[Gip10] Bela Gipp. Measuring Document Relatedness by Citation Proximity Analysis and Citation Order Analysis. In *ECDL*, 2010.

[GJ79] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[GL96] Gene H. Golub and Charles F. Van Loan. *Matrix computations (3rd ed.)*. John Hopkins University Press, 1996.

[Has05] Mehdi Hassani. Approximation of the Lambert W Function. *RGMIA Research Report Collection*, 8, 2005.

[Hav03] Taher H. Haveliwala. Topic-Sensitive PageRank: A Context-Sensitive Ranking Algorithm for Web Search. *IEEE Trans. Knowl. Data Eng.*, 15(4):784–796, 2003.

[HFLC10] Guoming He, Haijun Feng, Cuiping Li and Hong Chen. Parallel SimRank computation on large graphs with iterative aggregation. In *KDD*, 2010.

[HHP06] Heasoo Hwang, Vagelis Hristidis and Yannis Papakonstantinou. ObjectRank: a system for authority-based search on databases. In *SIGMOD Conference*, pages 796–798, 2006.

[HJ90] Roger A. Horn and Charles R. Johnson. *Matrix Analysis*. Cambridge University Press, 1990.

[HLC+12] Guoming He, Cuiping Li, Hong Chen, Xiaoyong Du and Haijun Feng. Using Graphics Processors for High Performance SimRank Computation. *IEEE Trans. Knowl. Data Eng.*, 24(9):1711–1725, 2012.

[Jar07] Bo Jarneving. Bibliographic coupling and its application to research-front and other core documents. *J. Informetrics*, 1(4):287–307, 2007.

[JLH11] Ruoming Jin, Victor E. Lee and Hui Hong. Axiomatic ranking of network role similarity. In *KDD*, 2011.

[JW02] Glen Jeh and Jennifer Widom. SimRank: A measure of structural-context similarity. In *KDD*, 2002.

[Kes63a] M. M. Kessler. Bibliographic coupling between scientific papers. *Amer. Doc.*, 14(1):10–25, 1963.

[Kes63b] Maxwell Mirton Kessler. Bibliographic coupling extended in time: Ten case histories. *Information Storage and Retrieval*, 1(4):169–187, 1963.

[Kes63c] Maxwell Mirton Kessler. An experimental study of bibliographic coupling between technical papers. *IEEE Transactions on Information Theory*, 9(1):49–51, 1963.

[Kle99] Jon M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. *J. ACM*, 46(5):604–632, 1999.

[KNT06] Ravi Kumar, Jasmine Novak and Andrew Tomkins. Structure and evolution of online social networks. In *KDD*, 2006.

[KSJ09] Ioannis Konstas, Vassilios Stathopoulos and Joemon M. Jose. On social networks and collaborative recommendation. In *SIGIR*, pages 195–202, 2009.

[Lau04] Alan J. Laub. *Matrix Analysis for Scientists and Engineers*. SIAM: Society for Industrial and Applied Mathematics, 2004.

[LHH+10] Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu and Tianyi Wu. Fast computation of SimRank for static and dynamic information networks. In *EDBT*, 2010.

[LHK10] Jure Leskovec, Daniel P. Huttenlocher and Jon M. Kleinberg. Signed networks in social media. In *CHI*, pages 1361–1370, 2010.

[LHN06] E. A. Leicht, Petter Holme and M. E. J. Newman. Vertex similarity in networks. *Physical Review E*, 73(2), 2006.

[Lin00] Xuemin Lin. On the computational complexity of edge concentration. *Discrete Applied Mathematics*, 101(1-3):197–205, 2000.

[LLK12] Zhenjiang Lin, Michael R. Lyu and Irwin King. MatchSim: A novel similarity measure based on maximum neighborhood matching. *Knowl. Inf. Syst.*, 32(1), 2012.

[LLY+10] Pei Li, Hongyan Liu, Jeffrey Xu Yu, Jun He and Xiaoyong Du. Fast Single-Pair SimRank Computation. In *SDM*, 2010.

[LLY12] Pei Lee, Laks V. S. Lakshmanan and Jeffrey Xu Yu. On Top-k Structural Similarity Search. In *ICDE*, 2012.

[LM03] Amy Nicole Langville and Carl Dean Meyer. Survey: Deeper Inside PageRank. *Internet Mathematics*, 1(3):335–380, 2003.

[LNK03] David Liben-Nowell and Jon M. Kleinberg. The link prediction problem for social networks. In *CIKM*, pages 556–559, 2003.

[LVGT08] Dmitry Lizorkin, Pavel Velikhov, Maxim N. Grinev and Denis Turdakov. Accuracy estimate and optimization techniques for SimRank computation. *PVLDB*, 1:422–433, 2008.

[LVGT10] Dmitry Lizorkin, Pavel Velikhov, Maxim N. Grinev and Denis Turdakov. Accuracy estimate and optimization techniques for SimRank computation. *VLDB J.*, 19(1), 2010.

[McC90] Katherine W. McCain. Mapping authors in intellectual space: A technical overview. *Journal of the American Society for Information Science*, 41:433–443, 1990.

[Mey01] Carl Meyer. *Matrix Analysis and Applied Linear Algebra.* SIAM, Februar 2001.

[NCO04] Alexandros Ntoulas, Junghoo Cho and Christopher Olston. What's new on the web?: The evolution of the web from a search engine perspective. In *WWW*, 2004.

[PBMW99] Lawrence Page, Sergey Brin, Rajeev Motwani and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical report, Stanford InfoLab, November 1999.

[PCD+08] Josiane Xavier Parreira, Carlos Castillo, Debora Donato, Sebastian Michel and Gerhard Weikum. The Juxtaposed approximate PageRank method for robust PageRank approximation in a peer-to-peer web search network. *VLDB J.*, 17(2):291–313, 2008.

[PYFD04] Jiayu Pan, Hyung-Jeong Yang, Christos Faloutsos and Pinar Duygulu. Automatic multimedia cross-modal correlation discovery. In *KDD*, pages 653–658, 2004.

[Saa03] Yousef Saad. *Iterative Methods for Sparse Linear Systems, Second Edition.* Society for Industrial and Applied Mathematics, 2. Auflage, April 2003.

[SBC+06] Tamás Sarlós, András A. Benczúr, Károly Csalogány, Dániel Fogaras and Balázs Rácz. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *WWW*, pages 297–306, 2006.

[SBC+10] Rossano Schifanella, Alain Barrat, Ciro Cattuto, Benjamin Markines and Filippo Menczer. Folks in Folksonomies: social link prediction from shared metadata. In *WSDM*, pages 271–280, 2010.

[SGP11] Atish Das Sarma, Sreenivas Gollapudi and Rina Panigrahy. Estimating PageRank on graph streams. *J. ACM*, 58:13, 2011.

[SHY+11] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu and Tianyi Wu. PathSim: Meta Path-Based Top-k Similarity Search in Heterogeneous Information Networks. *PVLDB*, 4(11):992–1003, 2011.

[SHZ+09] Yizhou Sun, Jiawei Han, Peixiang Zhao, Zhijun Yin, Hong Cheng and Tianyi Wu. RankClus: Integrating clustering with ranking for heterogeneous information network analysis. In *EDBT*, pages 565–576, 2009.

[Sma73] Henry Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *J. Am. Soc. Inf. Sci.*, 24(4), 1973.

[SMP08] Purnamrita Sarkar, Andrew W. Moore and Amit Prakash. Fast incremental proximity search in large graphs. In *ICML*, pages 896–903, 2008.

[TFP06] Hanghang Tong, Christos Faloutsos and Jia-Yu Pan. Fast Random Walk with Restart and Its Applications. In *ICDM*, 2006.

[TFP08] Hanghang Tong, Christos Faloutsos and Jia-Yu Pan. Random walk with restart: Fast solutions and applications. *Knowl. Inf. Syst.*, 14(3):327–346, 2008.

[TJ13] Geng Tian and Liping Jing. Recommending scientific articles using bi-relational graph-based iterative RWR. In *RecSys*, pages 399–402, 2013.

[TKF09] Hanghang Tong, Yehuda Koren and Christos Faloutsos. Direction-Aware Proximity on Graphs. In *Encyclopedia of Data Warehousing and Mining*, pages 646–653. 2009.

[WG81] H. D. White and B. C. Griffith. Author co-citation: A literature measure of intellectual structure. *Journal of the American Society for Information Science*, 32:163–171, 1981.

[Wil07] Gareth Williams. *Linear Algebra with Applications*. Jones and Bartlett Publishers, 2007.

[WJZZ06] Changhu Wang, Feng Jing, Lei Zhang and HongJiang Zhang. Image annotation refinement using random walk with restarts. In *ACM Multimedia*, pages 647–650, 2006.

[XFF+05] Wensi Xi, Edward A. Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan and Dong Zhuang. SimFusion: Measuring similarity using unified relationship matrix. In *SIGIR*, 2005.

[XZC+04a] Wensi Xi, Benyu Zhang, Zheng Chen, Yizhou Lu, Shuicheng Yan, Weiying Ma and Edward A. Fox. Link Fusion: A unified link analysis framework for multi-type interrelated data objects. In *WWW*, pages 319–327, 2004.

[XZC+04b] Guirong Xue, Huajun Zeng, Zheng Chen, Yong Yu, Weiying Ma, Wensi Xi and Edward A. Fox. MRSSA: An iterative algorithm for similarity spreading over interrelated objects. In *CIKM*, pages 240–241, 2004.

[YHY06] Xiaoxin Yin, Jiawei Han and Philip S. Yu. LinkClus: Efficient Clustering via Heterogeneous Semantic Links. In *VLDB*, 2006.

[YLL10] Weiren Yu, Xuemin Lin and Jiajin Le. Taming Computational Complexity: Efficient and Parallel SimRank Optimizations on Undirected Graphs. In *WAIM*, 2010.

[YLZ+12] Weiren Yu, Xuemin Lin, Wenjie Zhang, Ying Zhang and Jiajin Le. SimFusion+: Extending SimFusion towards efficient estimation on large and dynamic networks. In *SIGIR*, 2012.

[YLZ13a] Weiren Yu, Xuemin Lin and Wenjie Zhang. Towards efficient SimRank computation on large networks. In *ICDE*, pages 601–612, 2013.

[YLZ+13b] Weiren Yu, Xuemin Lin, Wenjie Zhang, Lijun Chang and Jian Pei. More is Simpler: Effectively and Efficiently Assessing Node-Pair Similarities Based on Hyperlinks. http://www.cse.unsw.edu.au/~weirenyu/pubs/20130428.pdf UNSW-CSE-TR-201304, University of New South Wales, 2013.

[YMS14] Adams Wei Yu, Nikos Mamoulis and Hao Su. Reverse Top-k Search using Random Walk with Restart. *PVLDB*, 7(5):401–412, 2014.

[YZL+12] Weiren Yu, Wenjie Zhang, Xuemin Lin, Qing Zhang and Jiajin Le. A space and time efficient algorithm for SimRank computation. *World Wide Web*, 15(3):327–353, 2012.

[ZCY09] Yang Zhou, Hong Cheng and Jeffrey Xu Yu. Graph Clustering Based on Structural / Attribute Similarities. *PVLDB*, 2(1), 2009.

[ZFCY13] Fanwei Zhu, Yuan Fang, Kevin Chen-Chuan Chang and Jing Ying. Incremental and Accuracy-Aware Personalized PageRank through Scheduled Approximation. *PVLDB*, 6(6):481–492, 2013.

[ZHS09] Peixiang Zhao, Jiawei Han and Yizhou Sun. P-Rank: A comprehensive structural similarity measure over information networks. In *CIKM*, 2009.

[ZZF+13] Weiguo Zheng, Lei Zou, Yansong Feng, Lei Chen and Dongyan Zhao. Efficient SimRank-based Similarity Join Over Large Graphs. *PVLDB*, 6(7):493–504, 2013.