Efficient Partial-Pairs SimRank Search on Large Networks

Weiren Yu[†], Julie A. McCann[†]
[†]Imperial College London, United Kingdom
{weiren.yu, j.mccann}@imperial.ac.uk

ABSTRACT

The assessment of node-to-node similarities based on graph topology arises in a myriad of applications, *e.g.,* web search. SimRank is a notable measure of this type, with the intuition that "two nodes are similar if their in-neighbors are similar". While most existing work retrieving SimRank only considers *all-pairs* SimRank $s(\star,\star)$ and *single-source* SimRank $s(\star,j)$ (scores between every node and query j), there are appealing applications for *partial-pairs* SimRank, *e.g.,* similarity join. Given two node subsets A and B in a graph, partial-pairs SimRank assessment aims to retrieve only $\{s(a,b)\}_{\forall a\in A,\forall b\in B}$. However, the best-known solution appears not self-contained since it hinges on the premise that the SimRank scores with node-pairs in an h-go cover set must be given beforehand.

This paper focuses on efficient assessment of partial-pairs SimRank in a self-contained manner. (1) We devise a novel "seed germination" model that computes partial-pairs SimRank in $O(k|E|\min\{|A|,|B|\})$ time and $O(|E|+k|V|)$ memory for k iterations on a graph of $|V|$ nodes and $|E|$ edges. (2) We further eliminate unnecessary edge access to improve the time of partial-pairs SimRank to $O(m\min\{|A|,|B|\})$, where $m\leq\min\{k|E|,\Delta^{2k}\}$, and Δ is the maximum degree. (3) We show that our partial-pairs SimRank model also can handle the computations of all-pairs and single-source SimRanks. (4) We empirically verify that our algorithms are (a) 38x faster than the best-known competitors, and (b) memory-efficient, allowing scores to be assessed accurately on graphs with tens of millions of links.

1. INTRODUCTION

An overarching problem in link analysis is the measure of relevance between nodes based on graph topology. This type of relevance, termed *link-based similarity*, has long pervaded fertile fields of data management, *e.g.,* link prediction [7], web page ranking [2], and automatic image captioning [11]. Link-based similarity, unlike its text-based counterpart, is structure dependent and does not rely on domain knowledge. With the recurring emergence of many similarity measures, SimRank [7] has become attractive, due to its concise and recursive idea that "*two nodes are assessed as similar if their in-neighbors are similar*", along with the base case that "*every node is most similar to itself*". Compared with other link-based measures, SimRank bears two impressive merits: (1) It exploits global graph topology for both node-pair and node ranking, unlike PageRank [2] that only ranks nodes. (2) It also captures multifaceted relations (multiple paths of different lengths) between nodes, as opposed to Shortest Path that only tallies a single path of the shortest length. Hence, SimRank has been widely-accepted as an important tool for link analysis over the last decade [3–5, 7, 10, 16, 20].

However, prior iterative methods (*e.g.,* [13,16]) for assessing SimRank have two limitations:

(1) They mainly consider *all-pairs* SimRank $s(\star,\star)$ computation, and are not efficient if one needs to assess only *partial-pairs* similarities. This is because, even for retrieving a single-pair similarity $s_k(a,b)$ between nodes a and b at iteration k, their iterative models require all-pairs similarities $s_{k-1}(\star,\star)$ at iteration $(k-1)$ to be determined beforehand. We refer to this phenomenon as "*high iteration coupling*". To the best of our knowledge, there are several pioneering works trying to break the "high iteration coupling" barrier, such as *single-pair* SimRank [3, 6, 12], and top-K *single-source* SimRank [4,9,10], but they still require excessive time and memory, or deliver probabilistic results. (see Related Work section for a detailed comparison).

(2) Existing iterative methods for assessing all-pairs SimRank are not memory-efficient, which is the main obstacle to handling large graphs. Indeed, [13, 16] require $O(|V|^2)$ memory[1] at each iteration to store all-pairs scores from the previous iteration, which is due to "high iteration coupling". Thus, prior methods assessing all-pairs SimRank on a graph with $|V|\geq$ 500K, if without loss of exactness, would require $\geq(500\text{K})^2=250$G memory, which is rather impractical. To address this issue, Lizorkin *et al.* [13] devised an appealing heuristic to iteratively eliminate small similarities below a threshold. This method has the advantage of controlling the number of node-pairs of non-zero similarities fewer enough to fit into memory at the expense of exactness. However, it is difficult for the heuristic to enhance the quality of real applications. Fortunately, we observe that, if partial-pairs SimRank can break the "high iteration coupling" nature, all-pairs SimRank can be divided into several independent tasks of assessing partial-pairs SimRank with small memory.

Proceedings of the VLDB Endowment, Vol. 8, No. 5
Copyright 2015 VLDB Endowment 2150-8097/15/01.

[1] Though [13, 16] claimed their methods require $O(|V|)$ *intermediate* memory, they exclude $O(|V|^2)$ memory to write all-pairs similarity outputs from the previous iteration.

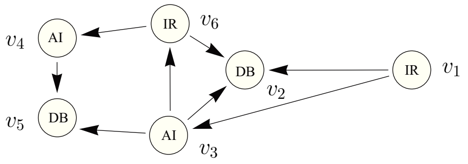


Figure 1: A Labeled Citation Network G

To resolve these issues, we consider the following problem.

Problem (PARTIAL-PAIRS SIMRANK ASSESSMENT)
Given a digraph $G = (V, E)$, a decay factor $C \in (0, 1)$, and two collections A and B of nodes in V.
Retrieve partial-pairs SimRank $\{s(a, b)\}_{\forall a \in A, \forall b \in B}$.

Partial-pairs assessment is useful in real applications. First, it is a unified model, containing as special cases the existing all-pairs [13,16], single-source [4,10], and single-pair [6,12] SimRank, when (A, B) is taken to be (V, V), $(V, \{j\})$, and $(\{i\}, \{j\})$, respectively, with queries i and j. Thus, it is applicable to any domains where all-pairs, single-source, and single-pair SimRank apply. Besides, assessing partial-pairs SimRank has its own applications, as depicted in Example 1.

EXAMPLE 1. *Figure 1 depicts a citation graph, where each node is a paper labeled with its relevant research areas, e.g., DB, IR, AI, and each edge $i \to j$ denotes a paper i cites j. A scientist, who is interested in interdisciplinary research of DB and AI, can utilize partial-pairs SimRank to assess only similarities of papers between DB and AI areas efficiently, without computing paper similarities outside these areas.* □

The best-known work [20] provides an excellent exposition of a threshold-based version that retrieves $\{(a, b)\}_{\forall a \in A, \forall b \in B}$ whose SimRank score $s(a, b)$ is above a given threshold θ. However, [20] is not self-contained as it relies on the premise that the SimRank scores with node-pairs in *an h-go cover*[2] must be known in advance. Precisely, for a given graph G, first, [20] adopts a heuristic (*i.e.,* Algorithms 2 and 3 of [20]) to find a portion of node-pairs on the tensor graph $G \otimes G$ (known as *an h-go cover set*). Then, the SimRank scores with node-pairs in this h-go cover set have to be provided as an input parameter T_B (in Algorithms 1 and 4 of [20]) and materialized, aiming to find the similarities of other node-pairs in $A \times B$ but outside the h-go cover set. However, it is hard to know in advance the SimRank scores of the h-go cover set (*i.e.,* the input parameter T_B of Algorithms 1 and 4). In addition, there is no rigorous complexity analysis in [20]. This highlights our need to focus on the efficiency of assessing partial-pairs SimRank in a self-contained manner.

There are two challenges of assessing partial-pairs scores. (1) Due to "high iteration coupling", it is difficult to break the holistic nature of the conventional iterative models for retrieving only partial-pairs SimRank in a self-contained style. (2) It is a grand challenge to improve both time and memory for partial-pairs SimRank without loss of accuracy, since even for computing a single-pair score, the prior works [6,12] have comparable complexities of all-pairs SimRank [13,16].

1.1 Contributions

In this paper, we make the following key contributions:

- We first design a fast "seed germination" iterative model to retrieve partial-pairs SimRank in a self-contained fashion, in $O(k|E|\min\{|A|, |B|\})$ time and $O(|E| + k|V|)$ memory for k iterations on a graph of $|V|$ nodes and $|E|$ edges, where $|A|$ and $|B|$ are the sizes of two query node subsets A and B, respectively. (Section 3.1)
- We then propose a pruning strategy, coupled with a hashing structure, to eliminate unnecessary computation with no loss of accuracy, which can further speed up the time of partial-pairs SimRank to $O(m \min\{|A|, |B|\})$, where $m \leq \min\{k|E|, \Delta^{2k}\}$ and d is the maximum degree. (Section 3.2)
- By setting $A = V$ and $B = \{j\}$, our partial-pairs SimRank, as a special case, also improves the time of single-source SimRank to $O(\min\{k|E|, \Delta^{2k}\})$. In comparison, the fastest known top-K version [10] needs $O(\Delta^{2k})$ time. (Section 4.1)
- As a by-product, we derive an $O(|E| + k|V|)$-memory efficient all-pairs SimRank algorithm, by dividing it into independent tasks of partial-pairs SimRank, whose total time, $O(\min\{k|E|, \Delta^{2k}\}|V|)$, is comparable to the best-known all-pairs SimRank [13,16], yet without using $O(|V|^2)$ memory to store scores from previous iterations. (Section 4.2)

The empirical evaluations on real and synthetic data verify that (a) our partial-pairs SimRank is self-contained, and runs drastically faster than the best-known competitors, and (b) our methods are memory-efficient, allowing SimRank being assessed accurately on large graphs with over 69M links.

1.2 Related Work

The problem of efficient SimRank computation has been well-studied, starting from the work of Jeh and Widom [7], and continuing with a long line of research, [3,4,6,10–13,15–17,19,20]. A comparison of our results (in terms of computational time, memory, and accuracy) to those in previous works is summarized in Table 1. We categorize these works as follows:

1.2.1 Partial-Pairs SimRank

The most relevant work to our research is proposed by Zheng *et al.* [20]. They gave (1) an estimated shortest-path bound for SimRank scores to prune unlikely node-pairs, and (2) heuristics to find an h-go cover set, and stored the SimRank scores of the h-go cover set, based on which, other scores can be computed easily. However, by using only the algorithms of [20], it is hard to determine in advance the scores of node-pairs in the h-go cover set. Our work differs from [20] in that our methods are more efficient and self-contained, needing no determination of extra SimRank scores in advance. We also analyze the complexity bound.

Sun *et al.* [14] provided a top-K link-based similarity join, with an e-function that generalizes PageRank and SimRank. They also obtained e-function bounds to prune unlikely nodes, requiring $O(k^2|E|)$ time to estimate the upper bound of a single-pair score. However, our methods can accurately compute $|A| \times |B|$ pairs of scores in $O(k|E|\min\{|A|, |B|\})$ time.

1.2.2 Single-Source SimRank

Lee *et al.* [10] proposed TopSim-SM via a random surfing process for top-K $s(\star, j)$ search in $O(\Delta^{2k})$ worst-case time. They also used heuristics to merge some repeated nodes, truncate low scores, and prioritize propagation. Their methods have the merits of fast speed when the top-K size is very

[2] *An h-go cover* of a graph is a set of nodes whose removal leaves the graph without simple paths longer than h. [20]

Paper	Algorithm	Type	Time	Memory	Accuracy	Notes
[20]	SJR	partial pairs	not given	$< O(\|V\|^2)$	C^k	hard to analyze its complexity because it only works when the SimRank scores in an h-go cover are given in advance)
This	PrunPar-SR		$O(\min\{k\|E\|, \Delta^{2k}\} \min\{\|A\|, \|B\|\})$	$O(\|E\| + k\|V\|)$	C^k	$\|A\|$ and $\|B\|$ are the query size of partial node-pair sets (A, B)
[3]	PSimRank	single pair	$O(kN)$	$O(\|E\| + N\|V\|)$	probabilistic	N is the number of sampling random walks via Monte Carlo
[12]	SingPair		$O(kd^2 \min\{\Delta^k, \|V\|^2\})$	$O(\|V\|^2)$	C^k	d is the graph average dgree; Δ is the graph maximum degree
[6]	ISP		$O(k\|E\|^2 - \|E\|)$	$O(\|V\|^2)$	C^k	an enhanced version of [12] based on position matrix
This	PrunPar-SR		$O(\min\{k\|E\|, \Delta^{2k}\})$	$O(\|E\| + k\|V\|)$	C^k	a special case of PrunPar-SR when $(A, B) := (\{i\}, \{j\})$
[4]	SimMat	single source	$O(r\|V\|^2 + r\|V\|)$	$O(r\|V\|^2)$	not given	r ($\leq \|V\|$) is the low rank of singular value decomposition
[9]	Monte-Carlo-SR		$O(k^2\|E\| + k(P + N)\|V\|)$	$O(\|E\| + P\|V\|)$	probabilistic	P is the iterations to build a node index via Monte Carlo
[10]	TopSim-SM		$O(\Delta^{2k})$	$O(\Delta^{2k} + \|V\|)$	C^k	complexity exponentially grows with the iteration number k
This	PrunPar-SR		$O(\min\{k\|E\|, \Delta^{2k}\})$	$O(\|E\| + k\|V\|)$	C^k	a special case of PrunPar-SR when $(A, B) := (V, \{j\})$
[7]	SimRank	all pairs	$O(kd^2\|V\|^2)$	$O(\|V\|^2)$	C^k	a naive algorithm via radius-based pruning
[11]	NI-Sim		$O(r^4\|V\|^2)$	$O(r^2\|V\|^2)$	not given	a non-iterative model via singular value deposition
[13]	Psum		$O(k \min\{\|V\|\|E\|, \frac{\|V\|^3}{\log_2 \|V\|}\})$	$O(\|V\|^2)$	C^k	needs $O(\|V\|^2)$ memory to store all the outputs for each iteration due to "high iteration coupling" barrier
[16]	OIP		$O(kd'\|V\|^2)$	$O(\|V\|^2)$	C^k	d' ($\leq \|E\|/\|V\|$) depends on common in-neighbours sharing
This	PrunPar-SR		$O(\min\{k\|E\|\|V\|, \Delta^{2k}\|V\|\})$	$O(\|E\| + k\|V\|)$	C^k	a special case of PrunPar-SR when $(A, B) := (V, V)$

Table 1: A comparison of our results to previous works in terms of computational time, memory usage, and accuracy

small, *e.g.,* K=50 ($\ll |V|$), but are not efficient for a larger top-K that will lead to more iterations k, since $O(\Delta^{2k})$ grows *exponentially w.r.t. k*. In contrast, our "seed germination" method, covering single-source SimRank as a special case, can avoid this exponential growth.

Kusumoto *et al.* [9] has proposed an algorithm via Monte Carlo method, called Monte-Carlo-SR, that retrieves top-K single-source SimRank in $O(k^2|E| + k(P + N)|V|)$ time, where P and N are the sampling parameters. Their method can avoid the exponential growth of the $O(\Delta^{2k})$ time in [10]. However, [9] delivers probabilistic results, and is slower than our "seed germination" model since it does not consider the computation sharing for single-source SimRank retrieval.

Recently, Fujiwara *et al.* [4] has devised an appealing matrix-based SimMat in $O(r|V|)$ time (exclude preprocessing time) to retrieve top-K $s(\star, j)$, where r is the low rank of approximation. Their approach is deterministic, but requires $O(r|V|^2)$ precomputation time to determine input matrices $\mathbf{L}$ and $\mathbf{R}$ for Algorithm 2 of [4], which is costly.

1.2.3 All-Pairs SimRank

Our focus is devoted to [11, 13, 16] as they are the state of the art. Lizorkin *et al.* [13] proposed an excellent method of partial sums memoization, reducing the time of all-pairs SimRank to $O(k|V||E|)$. They also devised a threshold-pruning heuristic and essential node-pairs selection for further improvement. Yu *et al.* [16] used a fine-grained clustering strategy for sub-summation sharing, yielding $O(kd'|V|^2)$ time with $d' \leq |E|/|V|$. Li *et al.* [11] used singular value decomposition for assessing SimRank in $O(r^4|V|^2)$ time, but it is not always efficient when the target rank r is not small. None of these methods are memory-efficient; [13,16] require $O(|V|^2)$ to store all-pairs scores from the previous iteration.

1.2.4 Single-Pair SimRank

The earliest mention of single-pair SimRank can be traced back to Fogaras *et al.* [3] who estimated $s(a, b)$ from the first hitting time of two random surfers starting from nodes a and b. However, [3] delivers *probabilistic* results. To enhance search quality, Li *et al.* [12] devised an iterative model in $O(kd^2 \min\{\Delta^k, |V|^2\})$ worst-case time and $O(|V|^2)$ memory, where d is the graph average degree. Recently, [6] has also employed position probability to reduce the time of [12] to $O(k|E|^2 - |E|)$. All the bounds of these deterministic single-pair SimRank are still expensive.

There has also been work [1, 8, 18] on SimRank variants. Antonellis *et al.* [1] extended SimRank for query rewriting. Jin *et al.* [8] integrated automorphism (role similarity) into SimRank. Yu *et al.* [18] devised SimRank*, by tallying more paths to enrich SimRank semantics. Our methods can also be readily ported to partial-pairs SimRank*. Due to space limitations, we omit the extension here.

2. PRELIMINARIES

Based on the idea that "two nodes are assessed as similar if their in-neighbors are similar", SimRank has two forms:

(1) Basic Form [7, 13]. Given a digraph $G = (V, E)$, for any node $a \in V$, we denote by $\mathcal{I}(a) := \{x \in V | \exists (x, a) \in E\}$ *the in-neighbor set of* a, and $|\mathcal{I}(a)|$ *the in-degree of* a.

The SimRank similarity between nodes a and b, $s(a, b)$, is defined as (i) $s(a, b) = 0$, if $\mathcal{I}(a)$ or $\mathcal{I}(b) = \varnothing$; (ii) otherwise,

$$s(a,b) = \begin{cases} 1, & a = b; \\ \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{(i,j) \in \mathcal{I}(a) \times \mathcal{I}(b)} s(i, j), & a \neq b. \end{cases} \quad (1)$$

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

(2) Matrix Form [5, 11]. Let $\mathbf{S} \in \mathbb{R}^{|V| \times |V|}$ be *a SimRank matrix* whose entry $[\mathbf{S}]_{i,j}$ is the score $s(i, j)$, $\mathbf{W}$ *the column normalized adjacency matrix* whose entry $[\mathbf{W}]_{i,j} = 1/|\mathcal{I}(j)|$ if $\exists$ an edge $i \to j$, and 0 otherwise, and $\mathbf{I}$ the $|V| \times |V|$ identity matrix. Then, SimRank can be also defined as

$$\mathbf{S} = C \cdot \mathbf{W}^T \cdot \mathbf{S} \cdot \mathbf{W} + (1 - C) \cdot \mathbf{I}, \quad (2)$$

where $(\star)^T$ is the matrix transpose. The term $(1 - C) \cdot \mathbf{I}$ in Eq.(2) ensures all the diagonal entries of $\mathbf{S}$ are maximal, which guarantees that every node is most similar to itself.

Note that, in the work of [9], $(1 - C) \cdot \mathbf{I}$ is superseded by a diagonal correction matrix $\mathbf{D}$, which is more precise. Our methods below, with only slight modifications, are also suitable for the model in [9].

3. OUR SOLUTIONS

Our solutions to partial-pairs SimRank involve two ideas: (1) We devise a novel "seed germination" iterative model that breaks "high iteration coupling" of the existing models.

(2) In light of this, we design a pruning strategy to further skip unnecessary computation, without loss of accuracy.

The following notations are used throughout this paper. (1) $\mathbf{e}_j$ is a $|V| \times 1$ vector of all 0s except for a 1 in j-th entry. (2) For matrix $\mathbf{X}$, (a) let $[\mathbf{X}]_{\star,j}$ be the j-th column of $\mathbf{X}$, (b) $[\mathbf{X}]_{i,\star}$ the i-th row of $\mathbf{X}$, (c) $[\mathbf{X}]_{i,j}$ the (i,j)-entry of $\mathbf{X}$.

3.1 A "Seed Germination" Iterative Model

We first formulate the existing "high iteration coupling" barriers, and then propose our "seed germination" methods.

3.1.1 *"High Iteration Coupling" Barrier*

Let $\mathbf{S}_k$ denote the k-th iterative SimRank matrix whose entry $[\mathbf{S}_k]_{i,j} = s_k(i,j)$. Then, $\mathbf{S}$ in Eq.(2) can be iterated as

$$\mathbf{S}_k = C \cdot \mathbf{W}^T \cdot \mathbf{S}_{k-1} \cdot \mathbf{W} + (1-C) \cdot \mathbf{I}. \tag{3}$$

By post-multiplying by $\mathbf{e}_j$ on both sides of Eq.(3), and using the fact that $\mathbf{X} \cdot \mathbf{e}_j = [\mathbf{X}]_{\star,j}$, it follows that

$$\underbrace{[\mathbf{S}_k]_{\star,j}}_{j\text{-th column}} = C \cdot \mathbf{W}^T \cdot \underbrace{\mathbf{S}_{k-1}}_{\text{entire matrix}} \cdot [\mathbf{W}]_{\star,j} + (1-C) \cdot \mathbf{e}_j, \quad \forall j. \tag{4}$$

Intuitively, Eq.(4) indicates that, even for finding only a single column $[\mathbf{S}_k]_{\star,j}$, the *entire* matrix $\mathbf{S}_{k-1}$ (not $[\mathbf{S}_{k-1}]_{\star,j}$) at iteration $(k-1)$ must be known in advance. Thus, it is hard for the conventional Eq.(3) to find only partial-pairs SimRank, without iteratively computing scores of others. We call this the SimRank *"high iteration coupling"* barrier.

3.1.2 *Partial Series Form of* $\mathbf{S}_k$

To break the barrier of "high iteration coupling", we first express $\mathbf{S}_k$ as a partial series form. This is because, from Eq.(4), we notice that the computational bottleneck of $[\mathbf{S}_k]_{\star,j}$ is the appearing of the *entire* matrix $\mathbf{S}_{k-1}$ on the right-hand side. To prevent its appearing, we have the following lemma.

Lemma 1. *For every $k = 0, 1, \cdots$, the k-th iterative SimRank matrix $\mathbf{S}_k$ derived from Eq.(3) can be expressed as*

$$\mathbf{S}_k = (1-C) \cdot \textstyle\sum_{l=0}^{k} C^l \cdot (\mathbf{W}^T)^l \cdot \mathbf{W}^l. \tag{5}$$

(The proof can be readily completed by induction on k, and is omitted here, due to space limitations.)

Lemma 1 tells us that $\mathbf{S}_k$ in Eq.(3) is actually the k-th partial sum of the infinite SimRank power series.

3.1.3 *Duplicate Computation in* $\mathbf{S}_k$

Lemma 1 is introduced for efficiently computing $[\mathbf{S}_k]_{\star,j}$. Unfortunately, Eq.(5), if carried out naively, contains many duplicate computations, as illustrated below.

First, we post-multiply by $\mathbf{e}_j$ on both sides of Eq.(5):

$$\begin{aligned} [\mathbf{S}_k]_{\star,j} &= (1-C) \textstyle\sum_{l=0}^{k} C^l \cdot (\mathbf{W}^T)^l \cdot \mathbf{W}^l \cdot \mathbf{e}_j \\ &= (1-C)\big(\mathbf{e}_j + \textstyle\sum_{l=1}^{k} C^l \cdot (\mathbf{W}^T)^l \cdot \mathbf{W}^{l-1} \cdot [\mathbf{W}]_{\star,j}\big) \end{aligned} \tag{6}$$

Since $[\mathbf{W}]_{\star,j}$ is a vector in Eq.(6), if the multiplications in $(\mathbf{W}^T)^l \cdot \mathbf{W}^{l-1} \cdot [\mathbf{W}]_{\star,j}$ are grouped from "right-to-left" as

$$\underbrace{\mathbf{W}^T \cdot \ldots \cdot (\mathbf{W}^T}_{l} \cdot (\underbrace{\mathbf{W} \cdot \ldots \cdot (\mathbf{W} \cdot (\mathbf{W}}_{l-1} \cdot [\mathbf{W}]_{\star,j})))),$$

then we can safely avoid using *matrix-matrix* multiplications while preventing the appearing of the entire matrix $\mathbf{S}_{k-1}$ throughout the summation of computing $[\mathbf{S}_k]_{\star,j}$ in Eq.(6). Specifically, to compute $(\mathbf{W}^T)^l \cdot \mathbf{W}^{l-1} \cdot [\mathbf{W}]_{\star,j}$, our "right-to-left association" needs $(2l-1)$ *matrix-vector* multiplications, whereas the traditional way entails (a) $(2l-2)$ *matrix-matrix* multiplications ($\mathbf{T} \leftarrow (\mathbf{W}^T)^l \cdot \mathbf{W}^{l-1}$) and (b) one matrix-vector multiplication ($\mathbf{T} \cdot [\mathbf{W}]_{\star,j}$), being rather expensive.

However, only the "right-to-left association" is not enough since there is duplicate computation across many summands. For example in Eq.(6), by writing out the terms for $l = 2, 3$:

$$\begin{aligned} l = 2 &: \underline{\mathbf{W}^T \cdot \mathbf{W}^T \cdot} \underline{\utilde{\mathbf{W}}} \utilde{\cdot [\mathbf{W}]_{\star,j}} \\ l = 3 &: \mathbf{W}^T \cdot \underline{\mathbf{W}^T \cdot \mathbf{W}^T \cdot \mathbf{W}} \cdot \utilde{\mathbf{W} \cdot [\mathbf{W}]_{\star,j}} \end{aligned}$$

we notice that there are many common parts shared by the summands of $l = 2$ and $l = 3$, such as $\mathbf{W} \cdot [\mathbf{W}]_{\star,j}$ (wavy underlined) and $\mathbf{W}^T \cdot \mathbf{W}^T \cdot \mathbf{W}$ (underlined). Unfortunately, it is hard to *maximally* share these common parts among the summands of different l while employ "right-to-left association" to prevent matrix-matrix multiplications at the same time, since some common parts sharing may destroy the "right-to-left association". As an example for $l = 2, 3$, our only opportunity seems to memoize the common parts $\mathbf{W} \cdot [\mathbf{W}]_{\star,j}$, in order to secure the "right-to-left association". Otherwise, if we memoized more common parts for sharing (*e.g.*, $\mathbf{W}^T \cdot \mathbf{W}^T \cdot \mathbf{W}$), *matrix-matrix* multiplications would be unnecessarily involved in Eq.(6) during $[\mathbf{S}_k]_{\star,j}$ computation.

3.1.4 *A "Seed Germination" Iterative Model for Single-Source SimRank*

To eliminate duplicate computation across the summands in Eq.(6) and also guarantee the "right-to-left association", we now propose our "seed germination" model.

Theorem 1. *Given a query j, the single-source SimRank $[\mathbf{S}_k]_{\star,j}$ at iteration k can be computed as*

$$[\mathbf{S}_k]_{\star,j} = (1-C) \cdot \mathbf{v}_k, \tag{7}$$

where vector $\mathbf{v}_k$ is iterated as: $\forall l = 1, \cdots, k$

$$\mathbf{v}_l = C \cdot \mathbf{W}^T \cdot \mathbf{v}_{l-1} + \mathbf{u}_{k-l} \quad \textit{with} \quad \mathbf{v}_0 = \mathbf{u}_k \tag{8}$$

after vectors $\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_k$ are iterated as: $\forall l = 1, \cdots, k$

$$\mathbf{u}_l = \mathbf{W} \cdot \mathbf{u}_{l-1} \quad \textit{with} \quad \mathbf{u}_0 = \mathbf{e}_j \tag{9}$$

Proof. We show that $[\mathbf{S}_k]_{\star,j}$ obtained from Eqs.(7)–(9) is exactly the k-th partial sum in Eq.(6).

First, successive substitution applied to Eq.(9) yields $\mathbf{u}_l = \mathbf{W}^l \cdot \mathbf{e}_j$. Thereby, $[\mathbf{S}_k]_{\star,j}$ in Eq.(6) can be rewritten as

$$[\mathbf{S}_k]_{\star,j} = (1-C) \cdot \textstyle\sum_{l=0}^{k} C^l \cdot (\mathbf{W}^T)^l \cdot \mathbf{u}_l \tag{10}$$

Next, we show that successive substitution applied to Eq.(8) yields $\mathbf{v}_k = \sum_{l=0}^{k} C^l \cdot (\mathbf{W}^T)^l \cdot \mathbf{u}_l$. Fixing l and pre-multiplying by $C^{k-l}(\mathbf{W}^T)^{k-l}$ on both sides of Eq.(8) produce

$$C^{k-l}(\mathbf{W}^T)^{k-l} \cdot \mathbf{v}_l = C^{k-l+1}(\mathbf{W}^T)^{k-l+1} \cdot \mathbf{v}_{l-1} + C^{k-l}(\mathbf{W}^T)^{k-l} \cdot \mathbf{u}_{k-l}$$

Then, taking sums $\sum_{l=1}^{k} (\star)$ on both sides yields

$$\underbrace{\sum_{l=1}^{k} C^{k-l} \cdot (\mathbf{W}^T)^{k-l} \cdot \mathbf{v}_l}_{= \mathbf{v}_k + \sum_{l=1}^{k-1} C^{k-l} \cdot (\mathbf{W}^T)^{k-l} \cdot \mathbf{v}_l} = \underbrace{\sum_{l=1}^{k} C^{k-l+1} \cdot (\mathbf{W}^T)^{k-l+1} \cdot \mathbf{v}_{l-1}}_{= \sum_{l=0}^{k-1} C^{k-l} \cdot (\mathbf{W}^T)^{k-l} \cdot \mathbf{v}_l} + \underbrace{\sum_{l=1}^{k} C^{k-l} \cdot (\mathbf{W}^T)^{k-l} \cdot \mathbf{u}_{k-l}}_{= \sum_{l=0}^{k-1} C^{l} \cdot (\mathbf{W}^T)^{l} \cdot \mathbf{u}_l}$$

Eliminating $\sum_{l=1}^{k-1} C^{k-l} \cdot (\mathbf{W}^T)^{k-l} \cdot \mathbf{v}_l$ on both sides gets

$$\mathbf{v}_k = C^k \cdot (\mathbf{W}^T)^k \cdot \mathbf{v}_0 + \textstyle\sum_{l=0}^{k-1} C^l \cdot (\mathbf{W}^T)^l \cdot \mathbf{u}_l = \sum_{l=0}^{k} C^l \cdot (\mathbf{W}^T)^l \cdot \mathbf{u}_l$$

Combining this with Eq.(10) yields Eq.(6). □

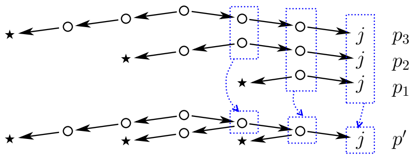


Figure 2: Merge paths p_1, p_2, p_3 to a compact tree p'

As an illustrative example, the last column of Figure 3 depicts how Theorem 1 computes $[\mathbf{S}_3]_{\star,j}$ step by step.

Theorem 1 provides a novel iterative algorithm for computing single-source SimRank $[\mathbf{S}_k]_{\star,j}$. It works as follows. Given query j and iteration number k, it first utilizes Eq.(9) to iteratively compute k auxiliary vectors $\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k$. Using $\mathbf{u}_1, \cdots, \mathbf{u}_k$, it then iteratively obtains $\mathbf{v}_k$ from Eq.(8). For each iteration l in Eq.(8), once $\mathbf{v}_l$ is computed, $\mathbf{v}_{l-1}$ and $\mathbf{u}_{k-l}$ can be freed. This iterative process continues until l reaches k. Finally, applying $\mathbf{v}_k$ to Eq.(7) returns $[\mathbf{S}_k]_{\star,j}$.

Theorem 1 can effectively skip duplicate multiplications across the summands in Eq.(6). Precisely, our method only requires $2k$ matrix-vector multiplications (k multiplications for $\mathbf{W} \cdot \mathbf{u}_{l-1}$ in Eq.(9) and k for $\mathbf{W}^T \cdot \mathbf{v}_{l-1}$ in Eq.(8)). In contrast, even though "right-to-left association" is used, Eq.(6) requires $(2l-1)$ matrix-vector multiplications for the l-th summand; thus, for k summands, Eq.(6) requires $\sum_{l=1}^{k}(2l-1) = k^2$ matrix-vector multiplications in total.

3.1.5 Main Idea

Intuitively, the idea of our method in Theorem 1 to eliminate computational redundancies is an efficient "seed germination" way of tallying "specific" paths for assessing $[\mathbf{S}_k]_{\star,j}$. To clarify this, we begin with the following lemma.

LEMMA 2. *The single-source $[\mathbf{S}_k]_{\star,j}$ in Eq.(6) tallies the weighted sum of k length-$2l$ "specific paths" ($l = 1, \cdots, k$):*

$$\underbrace{\overbrace{\star \leftarrow \circ \leftarrow \cdots \leftarrow \circ \leftarrow}^{l \text{ length}}}_{(\mathbf{W}^T)^l} \circ \underbrace{\overbrace{\rightarrow \circ \rightarrow \cdots \rightarrow}^{l-1 \text{ length}}}_{\mathbf{W}^{l-1}} \underbrace{\circ \rightarrow j}_{[\mathbf{W}]_{\circ,j}} \tag{11}$$

where $\circ$ and $\star$ denote any node in graph G.

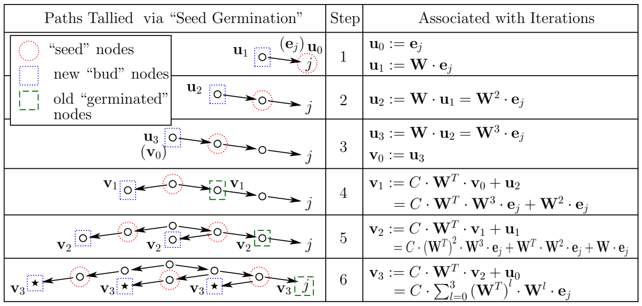
(The proof can be readily completed by the adjacency matrix power property. We omit it here, due to space limits.)

Lemma 2 suggests that assessing $[\mathbf{S}_k]_{\star,j}$ in Eq.(6) is actually the process of tallying the weighted sum of the "specific" paths in (11). Hence, the problem of eliminating duplicate computation in $[\mathbf{S}_k]_{\star,j}$ boils down to the merging of repetitive node access when tallying paths (11) of different length. Furthermore, the "right-to-left association" in Eq.(6), in the context of paths tallying, is associated with the order of node access in paths (11) from right end j to left end $\star$.

Based on Lemma 2, we describe the essence of our method in Theorem 1, with an example of assessing $[\mathbf{S}_3]_{\star,j}$.

The conventional Eq.(6) using only the "right-to-left association" would carry out the following calculation:

$$\begin{aligned}[\mathbf{S}_3]_{\star,j} = (1-C)\big(\mathbf{e}_j + &\overbrace{\mathbf{W}^T \cdot [\mathbf{W}]_{\star,j}}^{\text{path } p_1} + \overbrace{\mathbf{W}^T \cdot (\mathbf{W}^T \cdot (\mathbf{W} \cdot [\mathbf{W}]_{\star,j}))}^{\text{path } p_2} \\ &+ \underbrace{\mathbf{W}^T \cdot (\mathbf{W}^T \cdot (\mathbf{W}^T \cdot (\mathbf{W} \cdot (\mathbf{W} \cdot [\mathbf{W}]_{\star,j}))))}_{\text{path } p_3}\big),\end{aligned}$$



Paths Tallied via "Seed Germination"	Step	Associated with Iterations
	1	$\mathbf{u}_0 := \mathbf{e}_j$ $\mathbf{u}_1 := \mathbf{W} \cdot \mathbf{e}_j$
	2	$\mathbf{u}_2 := \mathbf{W} \cdot \mathbf{u}_1 = \mathbf{W}^2 \cdot \mathbf{e}_j$
	3	$\mathbf{u}_3 := \mathbf{W} \cdot \mathbf{u}_2 = \mathbf{W}^3 \cdot \mathbf{e}_j$ $\mathbf{v}_0 := \mathbf{u}_3$
	4	$\mathbf{v}_1 := C \cdot \mathbf{W}^T \cdot \mathbf{v}_0 + \mathbf{u}_2$ $= C \cdot \mathbf{W}^T \cdot \mathbf{W}^3 \cdot \mathbf{e}_j + \mathbf{W}^2 \cdot \mathbf{e}_j$
	5	$\mathbf{v}_2 := C \cdot \mathbf{W}^T \cdot \mathbf{v}_1 + \mathbf{u}_1$ $= C \cdot (\mathrm{W}^T)^2 \cdot \mathrm{W}^3 \cdot \mathrm{e}_j + \mathrm{W}^T \cdot \mathrm{W}^2 \cdot \mathrm{e}_j + \mathrm{W} \cdot \mathrm{e}_j$
	6	$\mathbf{v}_3 := C \cdot \mathbf{W}^T \cdot \mathbf{v}_2 + \mathbf{u}_0$ $= C \cdot \sum_{l=0}^{3} (\mathbf{W}^T)^l \cdot \mathbf{W}^l \cdot \mathbf{e}_j$

Figure 3: Example of assessing $[\mathbf{S}_3]_{\star,j}$ via "seed germination" iteration, by tallying paths in tree p'

where each summand corresponds to tallying a kind of "specific" paths p_i, as shown in Figure 2.[3] When tallying these paths from right end j to left end $\star$, we observe that there are many repetitive node access (enclosed with dotted lines). For instance, node j and its in-neighbors are repeatedly accessed when paths p_1, p_2, p_3 are tallied; the 2-hop in-neighbors of node j are repeatedly accessed when p_2 and p_3 are tallied. To eliminate redundancies, we first merge these paths into a compact tree p' (in Figure 2), and then efficiently access all nodes in tree p' via "seed germination" iteration.

Figure 3 visualizes our "seed germination" iteration step-by-step for tallying paths in p' for $[\mathbf{S}_3]_{\star,j}$, starting from j. Each step corresponds to one iteration, at which we need to select "seed" nodes to produce new "bud" nodes, aiming to minimize repetitive node access for tallying p_1, p_2, p_3 in p'. For example, in "Step 1" row of Figure 3, we select node j as the "seed" to produce its "bud" nodes ($\circ$). This process is associated with the iteration $\mathbf{u}_1 := \mathbf{W} \cdot \mathbf{e}_j$. Moreover, in each step, we can choose as "seed" nodes either the new "bud" nodes from the last step (*e.g.,* "seeds" in Steps 1–4), or the mixture of the new "bud" nodes and the old "germinated" nodes from several prior steps (*e.g.,* "seeds" in Steps 5–6). For example, in "Step 5" row, the selected "seed" nodes (encircled with red dotted line) consist of two parts: one is the new "bud" nodes from Step 4 (boxed with blue dotted line, associated with the term $(C \cdot \mathbf{W}^T \cdot \mathbf{v}_0)$ in $\mathbf{v}_1$); the other is the old "germinated" nodes from Step 2 (boxed with green dashed line, associated with the term $\mathbf{u}_2$ in $\mathbf{v}_1$), both of which are integrated into $\mathbf{v}_1$ in Step 4 and used as "seeds" in Step 5. In contrast, the conventional Eq.(6) tallies each of the paths p_1, p_2, p_3 in Figure 2 *independently*, by choosing only *one* "seed" to produce only *one* "bud" in *one* path p_i at each step, incurring excessive computational cost.

3.1.6 Complexity of Single-Source SimRank

We next analyze the complexity of single-source SimRank.

THEOREM 2. *Given a graph $G = (V, E)$ and query $j \in V$, it takes $O(k|E|)$ time and $O(|E|+k|V|)$ memory to assess the k-th iterative single-source SimRank $[\mathbf{S}_k]_{\star,j}$ via Theorem 1.*

PROOF. Assessing $[\mathbf{S}_k]_{\star,j}$ via Theorem 1 has three phases:

(1) Iteratively obtaining $\mathbf{u}_0, \mathbf{u}_1, \cdots, \mathbf{u}_k$ from Eq.(9). For any fixed l, it requires $O(|E|)$ time and $O(|V|)$ memory to

[3] p_1, p_2, p_3 in Figure 2 are right-aligned at node j for merging, corresponding to the "right-to-left association" computing.

compute $\mathbf{u}_l$, which is dominated by the matrix-vector multiplication $(\mathbf{W} \cdot \mathbf{u}_{l-1})$. Once computed, $\mathbf{u}_0, \cdots, \mathbf{u}_k$ are memoized, which will be reused in subsequent iterations Eq.(8). Thus, this phase requires $O(k|E|)$ time and $O(k|V|)$ memory for k iterations, plus $O(|E|)$ memory to store the graph.

(2) Iteratively computing $\mathbf{v}_k$ *via Eq.*(8). It takes $O(|E|)$ time and $O(|V|)$ memory to compute $\mathbf{v}_l$ per iteration, which is dominated by the matrix-vector multiplication $(\mathbf{W}^T \cdot \mathbf{v}_{l-1})$. Once $\mathbf{v}_l$ is derived, $\mathbf{v}_{l-1}$ and $\mathbf{u}_{k-l}$ are freed. Thus, this phase takes $O(k|E|)$ time and $O(|V|)$ memory for k iterations.

(3) Computing $[\mathbf{S}_k]_{\star,j}$ *from* $\mathbf{v}_k$ *via Eq.*(7). This phase requires $O(|V|)$ time and $O(|V|)$ memory for vector scaling.

Taking (1)–(3) together, the total complexity is bounded by $O(k|E|)$ time and $O(k|V| + |E|)$ memory. □

In Section 3.2, this complexity will be further improved.

3.1.7 Applying to Partial-Pairs SimRank

We now apply the "seed germination" model of Theorem 1 to partial-pairs SimRank.

Let us first introduce the following notations. Given two subsets A and B of nodes in V, for any $|V| \times |V|$ matrix $\mathbf{X}$, we denote by $[\mathbf{X}]_{A,B}$ the $|A| \times |B|$ submatrix of $\mathbf{X}$ that lies on the intersection of each row in A with each column in B; $[\mathbf{X}]_{A,\star}$ the $|A| \times |V|$ submatrix of $\mathbf{X}$ selecting all rows in A; $[\mathbf{X}]_{\star,B}$ the $|V| \times |B|$ submatrix of $\mathbf{X}$ selecting all columns in B. For instance, given subsets $A = \{1,3\}$ and $B = \{1,2,4\}$ of node set $V = \{1,2,3,4\}$, and SimRank matrix $\mathbf{S} \in \mathbb{R}^{4\times4}$, below is the partial-pairs SimRank matrix $[\mathbf{S}]_{A,B}$.

$$\mathbf{S} = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \!\!\! \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} \boxed{s_{11}} & \boxed{s_{12}} & s_{13} & \boxed{s_{14}} \\ s_{21} & s_{22} & s_{23} & s_{24} \\ \boxed{s_{31}} & \boxed{s_{32}} & s_{33} & \boxed{s_{34}} \\ s_{41} & s_{42} & s_{43} & s_{44} \end{bmatrix} \end{array} \quad \Rightarrow \quad [\mathbf{S}]_{A,B} = \begin{bmatrix} s_{11} & s_{12} & s_{14} \\ s_{31} & s_{32} & s_{34} \end{bmatrix} \qquad (12)$$

We now consider the "seed germination" iteration for partial-pairs SimRank. Recall that its counterpart for single-source SimRank is based on two main ideas: one is the "right-to-left association" trick that avoids matrix-matrix multiplications; the other is to skip duplicate computation across the summations of Eq.(6) by tallying compacted paths in a "seed germination" manner. Indeed, the latter idea can be readily ported to partial-pairs SimRank, whereas the former is not a natural extension since it is unclear whether the "right-to-left association" is still well suited for partial-pairs SimRank. To shed light on this, we first provide the following lemma.

Lemma 3. *Given two subsets A and B of a node set V, the partial-pairs SimRank matrix $[\mathbf{S}]_{A,B}$ can be obtained from the entire $|V| \times |V|$ matrix $\mathbf{S}$ via the transformation below:*

$$[\mathbf{S}]_{A,B} = [\mathbf{I}]_{A,\star} \cdot \mathbf{S} \cdot [\mathbf{I}]_{\star,B}. \qquad (13)$$

(The proof can be completed by direct entry-wise manipulation to Eq.(13). We omit it for space interests.)

Intuitively, to transform $\mathbf{S} \in \mathbb{R}^{|V|\times|V|}$ to $[\mathbf{S}]_{A,B} \in \mathbb{R}^{|A|\times|B|}$, Lemma 3 constructs $[\mathbf{I}]_{A,\star} \in \mathbb{R}^{|A|\times|V|}$ ("row-selector") and $[\mathbf{I}]_{\star,B} \in \mathbb{R}^{|V|\times|B|}$ ("column-selector") such that Eq.(13) holds. For example in Eq.(12), we find $\exists\ [\mathbf{I}]_{A,\star} = [\mathbf{e}_1 \,|\, \mathbf{e}_3]^T \in \mathbb{R}^{2\times4}$ and $[\mathbf{I}]_{\star,B} = [\mathbf{e}_1 \,|\, \mathbf{e}_2 \,|\, \mathbf{e}_4] \in \mathbb{R}^{4\times3}$ *s.t.* $[\mathbf{S}]_{A,B} = [\mathbf{I}]_{A,\star} \cdot \mathbf{S} \cdot [\mathbf{I}]_{\star,B}$.

Lemma 3 is introduced for efficiently computing $[\mathbf{S}]_{A,B}$. To obtain an Eq.(6)-like formula for partial-pairs SimRank, we first pre-multiply by $[\mathbf{I}]_{A,\star}$ and post-multiply by $[\mathbf{I}]_{\star,B}$ on both sides of Eq.(5), and then apply Lemma 3, which yields

$$[\mathbf{S}_k]_{A,B} = (1-C)\Big([\mathbf{I}]_{A,B} + \sum_{l=1}^{k} C^l [\mathbf{W}^T]_{A,\star} (\mathbf{W}^T)^{l-1} \mathbf{W}^{l-1} [\mathbf{W}]_{\star,B}\Big) \qquad (14)$$

However, unlike single-source SimRank, the "right-to-left association" trick to Eq.(6) may or may not apply to Eq.(14), depending on the sizes of node collections A and B, and the graph structure. This is because both $[\mathbf{W}^T]_{A,\star}$ and $[\mathbf{W}]_{\star,B}$ in Eq.(14) are rectangular matrices. As the matrix multiplication satisfies the associative law, for each fixed l in Eq.(14), the multiplication orders in $[\mathbf{W}^T]_{A,\star} (\mathbf{W}^T)^{l-1} \mathbf{W}^{l-1} [\mathbf{W}]_{\star,B}$ can be tactically adjusted to reduce its computational cost.

Lemma 4. *For every fixed l in the summation of Eq.*(14)*, $[\mathbf{W}^T]_{A,\star} \cdot (\mathbf{W}^T)^{l-1} \cdot \mathbf{W}^{l-1} \cdot [\mathbf{W}]_{\star,B}$ can be efficiently computed in $O\big(2(l-1) \cdot \min\{|A|,|B|\} \cdot |E| + \Delta|A||B|\big)$ worst-case time, by grouping all the matrix multiplications from "left-to-right" (resp. "right-to-left") if $|A| < |B|$ (resp. $|A| \geq |B|$).*

Proof. We shall consider the following three cases:

(1) If multiplications are grouped from "left-to-right" as

$$(((((([\mathbf{W}^T]_{A,\star} \cdot \underbrace{\mathbf{W}^T) \cdot \mathbf{W}^T) \cdot \ldots \cdot \mathbf{W}^T}_{l-1}) \cdot \underbrace{\mathbf{W}) \cdot \ldots \cdot \mathbf{W})}_{l-1} \cdot [\mathbf{W}]_{\star,B}$$

then the total time is bounded by $O(2(l-1)|A||E| + \Delta|A||B|)$.

(2) If multiplications are grouped from "right-to-left" as

$$[\mathbf{W}^T]_{A,\star} \cdot (\underbrace{\mathbf{W}^T \cdot \ldots \cdot (\mathbf{W}^T}_{l-1} \cdot (\underbrace{\mathbf{W} \cdot \ldots \cdot (\mathbf{W} \cdot (\mathbf{W}}_{l-1} \cdot [\mathbf{W}]_{\star,B})))))$$

then the total time is bounded by $O(2(l-1)|B||E| + \Delta|A||B|)$.

(3) If multiplications are grouped from "both ends" to meet at a contain position p" $(p = 2,3,\cdots,2(l-1))$ as

$$\underbrace{((([\mathbf{W}^T]_{A,\star} \cdot \mathbf{W}^T) \cdot \mathbf{W}^T) \cdot \ldots)}_{p \text{ terms}} \cdot \underbrace{(\cdot \ldots \cdot (\mathbf{W} \cdot (\mathbf{W} \cdot [\mathbf{W}]_{\star,B})))}_{(2l-p) \text{ terms}}$$

then it needs $O((p-1)|A||E| + (2l-p-1)|B||E| + |V||A||B|)$ time in total. Note that the last part requires $O(|V||A||B|)$ time (in lieu of $O(\Delta|A||B|)$) since either of the two remaining rectangular matrices may not be sparse.

Apart from the above cases, other grouping orders of matrix multiplications beyond our consideration (*e.g.*, grouping from a position $p \in [2, 2(l-1)]$ to two ends) would incur excessive costs, since at least $O(|V||E|)$ time is needed for every multiplication of two $|V| \times |V|$ matrices, which will dominate the total time.

Based on the above analysis, it suffices to show that the optimum time is achieved by Case (1) or Case (2). Indeed, the time of Case (3) can be written as $O(f(p))$, with

$$\begin{aligned} f(p) &= (p-1)|A||E| + (2l-p-1)|B||E| + |V||A||B| \\ &= p(|A| - |B|)|E| + z \qquad (\forall p = 2,3,\cdots,2(l-1)) \end{aligned}$$

and $z := (2l-1)|B| - |A| + |V||A||B|$. Thus, the minimum value of $f(p)$ occurs at $p = \begin{cases} 2, & \text{if } |A| \geq |B|; \\ 2(l-1), & \text{if } |A| < |B|. \end{cases}$

We can verify the time of Cases (1) and (2) is better than $O(f(2(l-1)))$ and $O(f(2))$, respectively. Thus, the optimum time cannot be achieved by Case (3), $\forall p \in [2, 2(l-1)]$. □

Lemma 4 implies that a *unidirectional* order of grouping all matrix multiplications in $[\mathbf{W}^T]_{A,\star} (\mathbf{W}^T)^{l-1} \mathbf{W}^{l-1} [\mathbf{W}]_{\star,B}$ can attain the optimum computational time. It comprises our aforementioned "right-to-left association" trick of single-source SimRank as a special case when $(A,B) := (V,\{j\})$.

Algorithm 1: Par-SR (G, C, k, A, B)

```
   Input  : a graph $G = (V, E)$, decay factor $C$, #-iteration $k$,
            two subsets $A$ and $B$ of nodes in $V$.
   Output: the partial-pairs similarities $[\mathbf{S}_k]_{A,B}$.
1  if $|A| < |B|$ then swap $A$ and $B$ ;
2  initialize the column normalized adjacency matrix $\mathbf{W}$ of $G$ ;
3  foreach $j \in B$ do
4      initialize $\mathbf{u}_0 := \mathbf{e}_j$ ;
5      for $l := 1, \cdots, k$ do
6          compute $[\mathbf{u}_l]_i := [\mathbf{W}]_{i,\star} \cdot \mathbf{u}_{l-1} \quad (\forall i \in V)$ ;
7      initialize $\mathbf{v}_0 := \mathbf{u}_k$ ;
8      for $l := 1, \cdots, k-1$ do
9          compute $[\mathbf{v}_l]_i := C \cdot [\mathbf{W}]_{\star,i}^T \cdot \mathbf{v}_{l-1} + [\mathbf{u}_{k-l}]_i \ (\forall i \in V)$;
10         free $\mathbf{v}_{l-1}$ and $\mathbf{u}_{k-l}$ ;
11     compute $[\mathbf{S}_k]_{A,j} := (1-C) \cdot (C \cdot [\mathbf{W}]_{\star,A}^T \mathbf{v}_{k-1} + [\mathbf{I}]_{A,j})$ ;
12     free $\mathbf{v}_{k-1}$ ;
13 if $A$ and $B$ were swapped then return $[\mathbf{S}_k]_{A,B}^T$ ;
14 else return $[\mathbf{S}_k]_{A,B}$ ;
```

In what follows, we can tacitly assume that $|A| \geq |B|$; otherwise, the given A and B can be swapped without affecting the results, due to SimRank symmetry ($s(a, b) = s(b, a)$). With this assumption, the *unidirectional* grouping order in Lemma 4 refers particularly to "right-to-left association", the same as the grouping order for single-source SimRank. As such, our method of tallying paths in a compact tree for single-source SimRank can be readily ported to partial-pairs SimRank, as illustrated in Theorem 3.

THEOREM 3. *Given two subsets A and B of nodes in V (we assume $|A| \geq |B|$ without loss of generality), the partial-pair SimRank $[\mathbf{S}_k]_{A,B}$ at iteration k can be computed as*

$$[\mathbf{S}_k]_{A,B} = (1 - C) \cdot (C \cdot [\mathbf{W}^T]_{A,\star} \cdot \mathbf{V}_{k-1} + \mathbf{I}_{A,B}) \qquad (15)$$

where $\mathbf{V}_{k-1} \in \mathbb{R}^{|V| \times |B|}$ is iterated as: $\forall l = 1, \cdots, k-1$

$$\mathbf{V}_l = C \cdot \mathbf{W}^T \cdot \mathbf{V}_{l-1} + \mathbf{U}_{k-l} \quad \textit{with } \mathbf{V}_0 = \mathbf{U}_k \qquad (16)$$

after $\mathbf{U}_0, \mathbf{U}_1, \cdots, \mathbf{U}_k \in \mathbb{R}^{|V| \times |B|}$ are iterated as: $\forall l = 1, \cdots, k$

$$\mathbf{U}_l = \mathbf{W} \cdot \mathbf{U}_{l-1} \quad \textit{with } \mathbf{U}_0 = \mathbf{I}_{\star,B} \qquad (17)$$

(The proof is similar to that of Theorem 1. We omit it here.)

Theorem 3 provides an algorithm, referred to as Par-SR, to iteratively assess partial-pairs SimRank $[\mathbf{S}_k]_{A,B}$ in a self-contained manner. One can easily verify by direct manipulation that Par-SR correctly computes partial-pairs SimRank.

3.1.8 Complexity of Partial-Pairs SimRank

We now analyze the complexity of Par-SR in Algorithm 1.

THEOREM 4. *Given a graph $G = (V, E)$ and two subsets of nodes A and B, Par-SR needs $O(k|E| \min\{|A|, |B|\})$ time and $O(k|V| + |E|)$ memory to compute $[\mathbf{S}_k]_{A,B}$.* [4]

PROOF. Without loss of generality, we shall only consider the case when $|A| \geq |B|$. For each $j \in B$, Par-SR consists of three phases: (a) compute $\mathbf{u}_0, \cdots, \mathbf{u}_k$ (Lines 5–6), (b) obtain $\mathbf{v}_l$ (Lines 8–10), and (c) assess $[\mathbf{S}_k]_{A,j}$ (Lines 11–12). In each phase, it can be verified that $O(k|E|)$, $O(k|E|)$ and $O(d|A|)$ time (*resp.* $O(k|V|)$, $O(k|V|)$ and $O(|V|)$ memory) are required. Thus, the total complexity is bounded by $O(k|B||E|)$ (*resp.* $O(k|A||E|)$) time and $O(k|V| + |E|)$ memory for $|A| \geq |B|$ (*resp.* $|A| < |B|$). □

It is worth mentioning that, for achieving high memory efficiency, for every column index $j \in B$, we first compute $[\mathbf{U}_0]_{\star,j}, \cdots, [\mathbf{U}_k]_{\star,j} \in \mathbb{R}^{|V| \times 1}$ in Eq.(16) as: $\forall l = 1, \cdots, k$

$$[\mathbf{U}_l]_{\star,j} = \mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j} \ \text{ with } [\mathbf{U}_0]_{\star,j} = \mathbf{e}_j \qquad (18)$$

Using *columns* $[\mathbf{U}_0]_{\star,j}, \cdots, [\mathbf{U}_k]_{\star,j}$ (not *matrices* $\mathbf{U}_0, \cdots, \mathbf{U}_k$), we then solve $[\mathbf{V}_l]_{\star,j}$ in Eq.(16) as: $\forall l = 1, \cdots, k-1$

$$[\mathbf{V}_l]_{\star,j} = C \cdot \mathbf{W}^T [\mathbf{V}_{l-1}]_{\star,j} + [\mathbf{U}_{k-l}]_{\star,j} \ \text{ with } [\mathbf{V}_0]_{\star,j} = [\mathbf{U}_k]_{\star,j} \qquad (19)$$

Once $[\mathbf{V}_l]_{\star,j}$ is computed, $[\mathbf{U}_0]_{\star,j}, \cdots, [\mathbf{U}_k]_{\star,j}$ are all freed. This process continues until each $j \in B$ is walked through. Compared with Eqs.(16) and (17), our trick in Eqs.(18) and (19) that converts matrix-matrix multiplications (*e.g.,* $\mathbf{W} \cdot \mathbf{U}_{l-1}$) into matrix-vector multiplications (*e.g.,* $\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j}$) has high memory efficiency.

3.1.9 Iteration Accuracy

The following theorem shows that our "seed germination" iterative model can achieve exactly the same accuracy as the conventional iterative counterparts (*e.g.,* [6, 7, 12, 13, 16, 20]).

THEOREM 5. *For any iteration number $k = 0, 1, \cdots$, the gap between the partial-pairs SimRank $[\mathbf{S}_k]_{A,B}$ in Algorithm 1 and the exact one $[\mathbf{S}]_{A,B}$ is bounded by*

$$\|[\mathbf{S}_k]_{A,B} - [\mathbf{S}]_{A,B}\|_{\max} \leq C^{k+1},$$

where $\|\mathbf{X}\|_{max} := \max_{i,j}\{|[\mathbf{X}]_{i,j}|\}$ is the entry-wise max norm.

PROOF. It follows from Eq.(14) that $\|[\mathbf{S}_k]_{A,B} - [\mathbf{S}]_{A,B}\|_{\max} \leq$ $(1-C) \sum_{l=k+1}^{\infty} C^l \underbrace{\|[\mathbf{W}^T]_{A,\star} (\mathbf{W}^T)^{l-1} \mathbf{W}^{l-1} [\mathbf{W}]_{\star,B}\|_{\max}}_{\leq 1} \leq C^{k+1}$. □

Theorem 5 implies that, for attaining a given accuracy ϵ, the total number of iterations for Par-SR is $k = \lceil \log_c \epsilon \rceil - 1$. In practice, since the decay factor $C = 0.6$ is a preferable setting [13], a typical value of $k = 9$ suffices to guarantee $\epsilon = 0.6^{9+1} = 0.006 < 0.01$ accurate to two decimal places.

Indeed, the effect of k is a speed-accuracy trade-off. The higher k indicates more complexity to compute summands of Eq.(14), but makes $[\mathbf{S}_k]_{A,B}$ even closer to the exact $[\mathbf{S}]_{A,B}$. This is consistent with our interpretation of the "seed germination" model in Figure 2, where we have discerned that the increasing of k implies more paths p_i to be merged into a compact tree p', leading to high accuracy yet more costs.

3.2 Eliminating Unnecessary Edge Access

After the "seed germination" iteration has been devised to assess partial-pairs SimRank in $O(k|E| \min\{|A|, |B|\})$ time, our pruning strategy in this section will further reduce the time to $O(m \min\{|A|, |B|\})$, with $m \leq \min\{k|E|, \Delta^{2k}\}$.

3.2.1 *Traditional* SpMxM *Redundancy*

Recall that the computational time of Par-SR is dominated by the matrix multiplications ($\mathbf{W} \cdot \mathbf{U}_{l-1}$) in Eq.(17) and ($\mathbf{W}^T \cdot \mathbf{V}_{l-1}$) in Eq.(16). In Section 3.1, we simply adopt

[4] In the next section, we shall further improve the complexity of Par-SR via a pruning strategy, without loss of exactness.

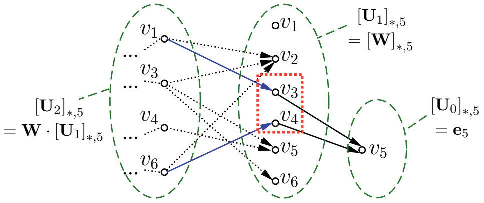


Figure 4: Edges Accessed for $[\mathbf{U}_2]_{\star,5} := \mathbf{W} \cdot [\mathbf{U}_1]_{\star,5}$ Computation during "Seed Germination" Iterations

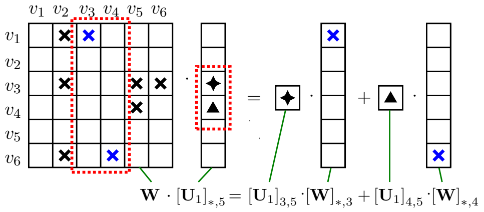

v_1 v_2 v_3 v_4 v_5 v_6
v_1
v_2
v_3
v_4
v_5
v_6
$\mathbf{W} \cdot [\mathbf{U}_1]_{*,5} = [\mathbf{U}_1]_{3,5} \cdot [\mathbf{W}]_{*,3} + [\mathbf{U}_1]_{4,5} \cdot [\mathbf{W}]_{*,4}$

Figure 5: Eliminating redundant multiplications by using Eq.(21) to express $\mathbf{W} \cdot [\mathbf{U}_1]_{\star,5}$ as combinations of only two columns ($[\mathbf{W}]_{\star,3}$ and $[\mathbf{W}]_{\star,4}$) of $\mathbf{W}$

standard sparse matrix multiplications (SpMxM), by viewing $\mathbf{W} \in \mathbb{R}^{|V| \times |V|}$ as a sparse matrix with $|E|$ nonzeros.[5] However, there exists unnecessary edge access in SpMxM.

To clarify this, let us interpret SpMxM in the context of our "seed germination" paradigm.[6] When SpMxM is used to compute $\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j}$, it can be written entry-wisely as

$$[\mathbf{U}_l]_{i,j} = \sum\nolimits_{y \in \mathcal{O}(i)} [\mathbf{W}]_{i,y} \cdot [\mathbf{U}_{l-1}]_{y,j} \text{ with } [\mathbf{U}_0]_{i,j} = \left\{\begin{smallmatrix} 1, & i=j \\ 0, & i \neq j \end{smallmatrix}\right., \quad \forall i,j \quad (20)$$

where $\mathcal{O}(i)$ is the out-neighbor set of node i. Consequently, $\sum_{i \in V} |\mathcal{O}(i)| = |E|$ edges are accessed to compute $\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j}$ via SpMxM, which contains redundancy in general.

EXAMPLE 2. *Recall graph G in Figure 1, and our iterative process of finding its $[\mathbf{S}]_{A,B}$ with $A = \{3,4\}$ and $B = \{2,5\}$. According to Theorem 3, when SpMxM is directly applied to compute e.g., $[\mathbf{U}_2]_{\star,5} := \mathbf{W} \cdot [\mathbf{U}_1]_{\star,5}$ with $[\mathbf{U}_1]_{\star,5} := [\mathbf{W}]_{\star,5}$, Figure 4 depicts the corresponding edges accessed by SpMxM.*

Specifically, SpMxM computes $(\mathbf{W} \cdot [\mathbf{U}_1]_{\star,5})$ by viewing $\mathbf{W}$ as a sparse matrix; the scanning of all nonzero entries at each nonzero row of $\mathbf{W}$ (i.e., $[\mathbf{W}]_{1,\star}, [\mathbf{W}]_{3,\star}, [\mathbf{W}]_{4,\star}, [\mathbf{W}]_{6,\star}$ in Figure 5) is equivalent to the access of all out-links (with dashed and blue solid arrows) of each node (i.e., v_1, v_3, v_4, v_6), respectively. Consequently, SpMxM requires $|E| = 8$ edges access for computing $(\mathbf{W} \cdot [\mathbf{U}_1]_{\star,5})$.

However, we observe that only 2 (with solid blue arrows) out of $|E| = 8$ edges are the useful access with contributions to scores $[\mathbf{S}_2]_{\star,5}$, as they are the essential parts that forms the path $\circ \to \circ \to v_5$ tallied by our "seed germination" iterations; other edges accessed in dashed arrows (associated with nonzeros in $\mathbf{W}$ except $[\mathbf{W}]_{1,3}$ and $[\mathbf{W}]_{6,4}$) are redundant as they cannot make $\circ \to \circ \to v_5$ contribute to $[\mathbf{S}_2]_{\star,5}$. □

Example 2 suggests unnecessary edge access involved in SpMxM. Indeed, such redundancies are often serious in real assessment, as evidenced by our experiments in Section 5, *e.g.*, nearly 38% edges of DBLP are redundant access. This hampers the efficiency of our partial-pairs assessment.

3.2.2 Pruning Redundant Edge Access

To eliminate unnecessary edge access of SpMxM for partial-pairs assessment, our main idea of computing $[\mathbf{U}_l]_{\star,j}$ in Eq.(18) is to express $(\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j})$ as combinations of columns of $\mathbf{W}$ as follows:

$$\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j} = \sum\nolimits_{x \in \mathcal{D}_j} \alpha_x \cdot [\mathbf{W}]_{\star,x} \quad \text{with } \alpha_i := [\mathbf{U}_{l-1}]_{i,j}$$
$$\text{and } \mathcal{D}_j := \{x \in V \mid [\mathbf{U}_{l-1}]_{x,j} \neq 0 \text{ and } \mathcal{I}(x) \neq \varnothing\} \quad (21)$$

The benefit of using this expression to compute $(\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j})$ is that we can effectively skip unnecessary multiplications in SpMxM, by pruning off a set of edges that are not germinated from query node j (called *"wild edges"*). Indeed, we observe that the "wild edges" accessed for computing $(\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j})$ are the incoming edges of node $x \in V \backslash \mathcal{D}_j$ whose $[\mathbf{U}_{l-1}]_{x,j}$ is zero, since successive substitution applied to Eq.(18) yields $[\mathbf{U}_{l-1}]_{\star,j} = \mathbf{W}^{l-1} \cdot \mathbf{e}_j = [\mathbf{W}^{l-1}]_{\star,j}$, which indicates that $[\mathbf{U}_{l-1}]_{x,j}$ (*i.e.*, $[\mathbf{W}^{l-1}]_{x,j}$) tallies the weights of length-$(l-1)$ paths from node x to query j. Thus, there are no length-$(l-1)$ paths from x to j whenever $[\mathbf{U}_{l-1}]_{x,j} = 0$. This tells us that all in-links of node $x \in V \backslash \mathcal{D}_j$ are not originally germinated from query j with l steps; they are "wild edges" that can be safely pruned when $(\mathbf{W} \cdot [\mathbf{U}_{l-1}]_{\star,j})$ is computed.

EXAMPLE 3. *Recall Example 2 and the iterative process in Figure 5: $[\mathbf{U}_2]_{\star,5} := \mathbf{W} \cdot [\mathbf{U}_1]_{\star,5}$ with $[\mathbf{U}_1]_{\star,5} := [\mathbf{W}]_{\star,5}$. Using our method of Eq.(21) to compute $(\mathbf{W} \cdot [\mathbf{U}_1]_{\star,5})$ needs only 2 edges access (denoted as blue solid arrow in Figure 4), corresponding to 2 multiplications, i.e.,*

$$\alpha_3 \cdot [\mathbf{W}]_{\star,3} + \alpha_4 \cdot [\mathbf{W}]_{\star,4} \textit{ with } \alpha_3 := [\mathbf{U}_1]_{3,5}, \quad \alpha_4 := [\mathbf{U}_1]_{4,5},$$

as pictured in Figure 5, where symbols ✖, ✦, ▲ in a square denote nonzero entries. Precisely, given $[\mathbf{U}_1]_{\star,5} := [\mathbf{W}]_{\star,5}$, Eq.(21) first finds $\mathcal{D}_5 := \{x \in V \mid [\mathbf{U}_1]_{x,5} \neq 0\} = \{v_3, v_4\}$ (framed in red dotted line in Figure 5), implying that all the in-links of node $x \notin \mathcal{D}_5$ are "wild edges" (in dashed arrows) that are not originally germinated from query v_5 with 2 steps. Then, Eq.(21) prunes off such "wild edges", by converting $(\mathbf{W} \cdot [\mathbf{U}_1]_{\star,5})$ to a combination of only two columns $[\mathbf{W}]_{\star,3}$ and $[\mathbf{W}]_{\star,4}$, with respective coefficients $[\mathbf{U}_1]_{3,5}$ and $[\mathbf{U}_1]_{4,5}$, as visualized in Figure 5. Thus, computing $(\mathbf{W} \cdot [\mathbf{U}_1]_{\star,5})$ via Eq.(21) needs only 2 multiplications, as opposed to SpMxM scanning all nonzeros of $\mathbf{W}$ with $|E| = 8$ multiplications. □

3.2.3 Hashing Strategy

To efficiently compute the combinations $\sum_{x \in \mathcal{D}_j} \alpha_x \cdot [\mathbf{W}]_{\star,x}$ in Eq.(21), besides using compressed column storage for $\mathbf{W}$, we also leverage a hashing strategy to eliminate unnecessary zero entry-wise vector additions. Precisely, we first build an empty hash table in such a way that the hash maps the node index i (hash key) to its value $[\mathbf{U}_l]_{i,j}$ (the result of $\sum_{x \in \mathcal{D}_j} \alpha_x \cdot [\mathbf{W}]_{i,x}$). To obtain $[\mathbf{U}_l]_{\star,j}$, for each $x \in \mathcal{D}_j$ we pick all nonzero entries of $[\mathbf{W}]_{\star,x}$ one by one, and search the hash table for the picked entry. If the entry exists in the table, we increase its value by $(\alpha_x \cdot [\mathbf{W}]_{i,x})$; otherwise, we enter it in the hash table with the initial value $(\alpha_x \cdot [\mathbf{W}]_{i,x})$. After all entries for $\forall x \in \mathcal{D}_j$ are entered in the hash table, the final result of $\sum_{x \in \mathcal{D}_j} \alpha_x \cdot [\mathbf{W}]_{\star,x}$ is derived. For example in Figure 5, our hashing method of computing $(\mathbf{W} \cdot [\mathbf{U}_1]_{\star,5})$ via Eq.(21) only needs 2 operations to enter $[\mathbf{U}_1]_{3,5} \cdot [\mathbf{W}]_{\star,3}$ and $[\mathbf{U}_1]_{4,5} \cdot [\mathbf{W}]_{\star,4}$ in the hash table.

[5] Real graphs are often sparse with $|E| \ll |V|^2$ in practice.
[6] For space interests, our focus will be devoted to $(\mathbf{W} \cdot \mathbf{U}_{l-1})$ in Eq.(17), which also suits $(\mathbf{W}^T \cdot \mathbf{V}_{l-1})$ in Eq.(16).

```
Algorithm 2: PrunPar-SR $(G, C, k, A, B)$
   Input/Output: the same as Algorithm 1.
1-2 the same as Lines 1–2 of Algorithm 1 ;
3   foreach $j \in B$ do
4       initialize $\mathbf{u}_0 := \mathbf{e}_j$ ;
5       for $l := 1, \cdots, k$ do
6           set $\mathcal{D} := \{x \in V \mid [\mathbf{u}_{l-1}]_x \neq 0$ and $\mathcal{I}(x) \neq \varnothing\}$ ;
7           $\mathbf{u}_l := \sum_{x \in \mathcal{D}} [\mathbf{u}_{l-1}]_x \cdot [\mathbf{W}]_{\star,x}$ via hashing ;
8       initialize $\mathbf{v}_0 := \mathbf{u}_k$ ;
9       for $l := 1, \cdots, k-1$ do
10          set $\mathcal{D} := \{x \in V \mid [\mathbf{v}_{l-1}]_x \neq 0$ and $\mathcal{O}(x) \neq \varnothing\}$ ;
11          $\mathbf{v}_l := C \cdot \sum_{x \in \mathcal{D}} [\mathbf{v}_{l-1}]_x \cdot [\mathbf{W}]^T_{x,\star} + \mathbf{u}_{k-l}$ via hashing ;
12          free $\mathbf{v}_{l-1}$ and $\mathbf{u}_{k-l}$ ;
13      set $\mathcal{D} := \{x \in V \mid [\mathbf{v}_{k-1}]_x \neq 0$ and $\mathcal{O}(x) \neq \varnothing\}$ ;
14      $[\mathbf{S}_k]_{A,j} := (1-C) \cdot (C \cdot \sum_{x \in \mathcal{D}} [\mathbf{v}_{k-1}]_x [\mathbf{W}]^T_{x,A} + [\mathbf{I}]_{A,j})$
            via hashing ;
15      free $\mathbf{v}_{k-1}$ ;
16-17 the same as Lines 13–14 of Algorithm 1 ;
```

3.2.4 Pruning Algorithm

By integrating our pruning and hashing methods into the partial-pairs SimRank, we now provide an enhanced version of Par-SR, referred to as PrunPar-SR, in Algorithm 2.

One can readily verify, by direct manipulation and Eq.(21), that PrunPar-SR (1) correctly prunes redundant edge access with a-priori zero scores in $[\mathbf{S}_k]_{A,B}$, and (2) returns exactly the same accuracy as Par-SR.

Regarding its complexity, we have the following theorem.

THEOREM 6. *Given a graph $G = (V, E)$ and two subsets of nodes A and B, PrunPar-SR requires $O(m \min\{|A|, |B|\})$ worst-case time and $O(|E| + k|V|)$ memory for k iterations, with $m \leq \min\{k|E|, \Delta^{2k}\}$ and Δ the maximum degree of G.*

PROOF. We can readily verify that the total complexity of PrunPar-SR is dominated by Lines 6–7 and Lines 10–11.

First, we show that, for every l, Lines 6–7 requires $O(m_l)$ time in the worst case, with $m_l \leq \min\{|E|, \Delta^l\}$.

(i) On one hand, for each $x \in \mathcal{D}_j$, it takes $O(|\mathcal{I}(x)|)$ time to compute $(\alpha_x \cdot [\mathbf{W}]_{\star,x})$, where $|\mathcal{I}(x)|$ is the in-degree of x. Thus, the total time of Lines 6–7 is bounded by

$$O(\textstyle\sum_{x \in \mathcal{D}_j} |\mathcal{I}(x)|) \leq O(\sum_{x \in V} |\mathcal{I}(x)|) = O(|E|).$$

(ii) On the other hand, let $|\mathbf{u}_{l-1}|$ be #-nonzeros in $\mathbf{u}_{l-1}$. Then, computing $\mathbf{u}_l := \mathbf{W} \cdot \mathbf{u}_{l-1}$ via Line 7 implies that

$$|\mathbf{u}_l| \leq \min\{\Delta \cdot |\mathbf{u}_{l-1}|, |V|\} \quad \text{with } |\mathbf{u}_0| = 1. \qquad (22)$$

This is because $(\mathbf{W} \cdot \mathbf{u}_{l-1})$ can be viewed as linear combinations of $|\mathbf{u}_{l-1}|$ columns of $\mathbf{W}$; each column has at most Δ nonzero entries. Thus, in the worst case when the position of each nonzero in each column is different from one another, there are at most $\min\{\Delta \cdot |\mathbf{u}_{l-1}|, |V|\}$ nonzero entries in $\mathbf{u}_l$. Successive substitution applied to Eq.(22) yields

$$|\mathbf{u}_{l-1}| \leq \min\{\Delta^{l-1}, |V|\}.$$

Hence, the cost of computing $(\mathbf{W} \cdot \mathbf{u}_{l-1})$ via Line 7 requires linear combinations of Δ^{l-1} columns of $\mathbf{W}$ in the worst case. Since there are at most Δ nonzero entries (*i.e.,* Δ multiplications via hashing) in each column, the total time is $O(\Delta^l)$.

Taking (i) and (ii) together, Lines 6–7 require $O(m_l)$ total time in the worst case, with $m_l \leq \min\{|E|, \Delta^l\}$.

Similarly, we can show that, for every l, Lines 10–11 yield $O(\tilde{m}_l)$ time in the worst case, where $\tilde{m}_l \leq \min\{|E|, \Delta^{k+l}\}$.

This is because computing $\mathbf{v}_l$ via Line 11 can be viewed as linear combinations of $|\mathbf{v}_{l-1}|$ rows of $\mathbf{W}$, then plus $\mathbf{u}_{k-l}$. Since there are at most Δ nonzeros in each row of $\mathbf{W}$ and at most Δ^{k-l} nonzeros in $\mathbf{u}_{k-l}$, we have

$$|\mathbf{v}_l| \leq \min\{\Delta \cdot |\mathbf{v}_{l-1}| + \Delta^{k-l}, |V|\} \quad \text{with } |\mathbf{v}_0| \leq \min\{\Delta^k, |V|\}.$$

Successive substitution applied to this produces

$$|\mathbf{v}_{l-1}| \leq \min\{\Delta^{k-l+1} \cdot \tfrac{\Delta^{2l}-1}{\Delta^2-1}, |V|\}.$$

Hence, the cost of computing $\mathbf{v}_l$ via Line 11 requires linear combinations of $\Delta^{k-l+1} \cdot \frac{\Delta^{2l}-1}{\Delta^2-1}$ rows of $\mathbf{W}$ in the worst case, then plus $\mathbf{u}_{k-l}$. Since there are at most Δ nonzeros (*i.e.,* Δ multiplications via hashing) in each row, and Δ^{k-l} nonzeros in $\mathbf{u}_{k-l}$, the total time for computing $\mathbf{v}_l$ can be bounded by $O(\Delta \cdot \Delta^{k-l+1} \cdot \frac{\Delta^{2l}-1}{\Delta^2-1} + \Delta^{k-l}) = O(\Delta^{k+l})$ in the worst case.

Thus, for k iterations, PrunPar-SR takes $O(m)$ total time with $m = \sum_{l=0}^{k} \max\{\tilde{m}_l, m_l\} \leq \min\{k|E|, \Delta^{2k}\}$. □

It is worth mentioning that the $O(m \min\{|A|, |B|\})$ time of PrunPar-SR with $m \leq \min\{k|E|, \Delta^{2k}\}$, on *dense graphs*[7], has the same complexity bound of Par-SR. However, in practice, PrunPar-SR runs much faster than Par-SR, especially in the first several iterations. In Section 5, we shall further experimentally evaluate the speedup of PrunPar-SR.

4. EXTENSIONS

We next provide some other consequences of PrunPar-SR.

4.1 Improving Single-Source SimRank

For any query j in a graph $G = (V, E)$, by setting $A := V$ and $B := \{j\}$ in PrunPar-SR, we have the following corollary.

COROLLARY 1. *For any query j in a graph $G = (V, E)$, it is in $O(\min\{k|E|, \Delta^{2k}\})$ time and $O(|E| + k|V|)$ memory to compute single-source SimRank $s_k(\star, j)$ for k iterations.*

In contrast to the best-known single-source SimRank [4, 9, 10], Corollary 1 has the following advantages: (1) Unlike the $O(\Delta^{2k})$ time of [10] that increases *exponentially w.r.t. k*, our method scales well to $|E|$. (2) The Monte Carlo method of [9] delivers probabilistic results, whereas our algorithm is deterministic. (3) [4] needs $O(r|V|^2)$ preprocessing time for a low-rank factorization, but our method has no such costs.

4.2 Reducing Memory for All-Pairs SimRank

By setting $A := V$ and $B := V$, another variant of PrunPar-SR is a memory-efficient algorithm for all-pairs SimRank.

COROLLARY 2. *Given a graph $G = (V, E)$, all-pairs SimRank $s_k(\star, \star)$ can be computed in $O(\min\{k|E||V|, \Delta^{2k}|V|\})$ time and $O(|E| + k|V|)$ memory for k iterations.*

Compared with the best-known all-pairs SimRank [16] that takes $O(kd'|V|^2)$ time with $d' \leq |E|/|V|$, Corollary 2 achieves high computational efficiency. Moreover, it breaks the "high iteration coupling" barrier of the conventional SimRank that requires to store $O(|V|^2)$ scores from previous iterations. In addition, our method is easy to be parallelized since each column $[\mathbf{S}_k]_{\star,j}$ can be computed simultaneously.

[7] Here, *a dense graph* refers to a graph with $|E| = O(|V|^2)$.

5. EXPERIMENTS

Our experiments on real and synthetic data will evaluate (1) the time, memory and accuracy of our partial-pairs SimRank, and (2) the high efficiency of its extension to single-source and all-pairs SimRank, and SimRank* model [18].

5.1 Experimental Setting

We use both real and synthetic data in our experiments.

(1) Real Data. For efficiency evaluation, we use 6 graphs[8]:

(a) P2P, a Gnutella peer-to-peer file sharing network, where each node is a host labeled with the number of its connected hosts (categorized into *e.g.,* leaf, ultrapeer), and an edge a connection from one host to another in the graph topology.

(b) DBLP, a co-authorship DBLP graph, where a node is an author labeled with his expertise (*e.g.,* DB, IR) categorized by his conference publications. An edge is a co-authorship.

(c) WebS, a web graph from `stanford.edu`, where each node is a page, labeled with its "importance" ordered by the PageRank values that are split into 200 equal-sized buckets. Each directed edge is a hyperlink between them.

(d) AM, an Amazon product co-purchasing graph, where a node is a product labeled with product category, and rating. An edge links products a and b if a is co-purchased with b.

(e) CitP, a US patent network, in which a node is a patent labeled with grant date, country of first inventor, technological category. Edges are directed citations made by patents.

(f) SocL, a LiveJournal social network, in which nodes are members labeled with their communities, countries, and ages (crawled from `Tartu` site[9]), and edges are friendships.

The size $|G|(|V|,|E|)$ of the graphs is shown as follows.

Data	$\|G\|$ $(\|V\|,\|E\|)$	d	Data	$\|G\|$ $(\|V\|,\|E\|)$	d
P2P	27.1K (6.3K, 20.8K)	3.3	AM	3.8M (403K, 3.4M)	8.4
DBLP	49.5K (13.2K, 36.3K)	2.7	CitP	20.3M (3.8M, 16.5M)	4.4
WebS	2.6M (282K, 2.3M)	8.2	SocL	73.8M (4.8M, 69.0M)	14.2

(2) Synthetic Data. To produce synthetic network SYN, we use a generator GTgraph[10], controlled by $|V|$ and $|E|$.

(3) Query Generator. (a) For partial-pairs $\{s(a,b)\}_{\forall a \in A, \forall b \in B}$ assessment, we generate query pairs (A,B) as follows: For queries on real graphs, the labels are taken from the datasets; for synthetic graphs, we randomly sample A and B from V, controlled by their size $|A|$ and $|B|$. To ensure that the selected nodes in A and B can comprehensively cover a broad range of any possible queries, we first divide all nodes ordered by their density ($|E|/|V|$) into 10 equal-sized buckets, and then randomly sample $\lceil \frac{1}{10}|A| \rceil$ and $\lceil \frac{1}{10}|B| \rceil$ nodes, respectively, from each bucket. (b) For single-source $s(\star, j)$ assessment on both real and synthetic graphs, we randomly select query j from V in a similar way. For every experiment, the average performance is reported over all test queries.

(4) Algorithms. We implement the following, all in VC++.

Algorithm	Description	Type
PrunPar-SR	our algorithm in Sect. 3.2, with pruning	partial pairs
Par-SR	our algorithm in Sect. 3.1, without pruning	
PrunPar-SR*	variation of PrunPar-SR ported to SimRank*	
SJR	SimRank-based similarity join [20]	
TopSim-SM	top-K random walk based SimRank [10]	single source
SimMat	top-K matrix-based SimRank [4]	
Psum	partial sum memoization SimRank [13]	all pairs
OIP	fine-grained memoization SimRank [16]	
Psum-SR*	partial sum memoization SimRank* [18]	
Memo-SR*	edge concentration SimRank* [18]	

(5) Parameters. We set the following default parameters: (a) $C = 0.6$, the decay factor suggested in [13]. (b) $k = 10$, the number of iterations that ensures $s_k(\star,\star)$ accurate to 2 decimal places as $C^{k+1} = 0.6^{10+1} < 0.01$, according to [13].

(6) Accuracy Metric. For accuracy comparison, we adopt *Normalized Discounted Cumulative Gain (NDCG) at position p w.r.t. query j*, $\text{NDCG}_p(j) = \frac{1}{\text{IDCG}_p(j)} \sum_{i=1}^{p} \frac{2^{s(i,j)}-1}{\log_2(1+i)}$, where $s(i,j)$ is the similarity score between nodes i and j, and $\text{IDCG}_p(j)$ is a normalized factor ensuring the "true" NDCG ordering to be 1. Since all the compared algorithms come from SimRank and SimRank* families, we can choose all-pairs scores of [13] and [18] as their baselines.

All experiments are run with an Intel Core(TM) i7-4700MQ CPU @ 2.40GHz CPU and 32GB RAM, on Windows 7.

5.2 Experimental Results

5.2.1 Time Efficiency

We run algorithms PrunPar-SR, Par-SR, SJR, and SingPair on six real datasets. By randomly issuing different queries, we compare their total time of assessing partial-pairs SimRank. Figure 6a depicts the results. (1) On each dataset, PrunPar-SR and Par-SR are *always* faster than SJR and SingPair. This is because our "seed germination" iterative model can merge repeated node access into a compact tree, significantly reducing the cost in the summations. (2) On P2P and CitP, PrunPar-SR is almost 5x faster than Par-SR. This suggests that our pruning is powerful on small density graph, which is consistent with our analysis in Section 3.2. (3) When the data size becomes larger, the time of each algorithm increases in general. However, SJR will crash on large AM, CitP, SocL, due to its expensive cost for finding an h-go cover on the tensor graph. SingPair only survives on small P2P due to large memory. However, PrunPar-SR and Par-SR scale well to large graphs, which is consistent with our complexity analysis in Section 3.

Figure 6b compares the time of PrunPar-SR, Par-SR, SJR with respect to the different query pairs (A,B) on DBLP. The results show that (1) for various queries, PrunPar-SR consistently outperforms Par-SR, SJR. (*e.g.,* it is 2.4x, 39.6x faster than Par-SR, SJR, respectively, for query (DB, DM).) (2) In all cases, SJR exhibits the worst performance since it computes every node-pair score in (A,B) one by one without computation sharing, unlike our "seed germination" method that merges repetitive node access. (3) When the query size of (A,B) grows, the time for PrunPar-SR, Par-SR increases. This is in good agreement with our complexity analysis.

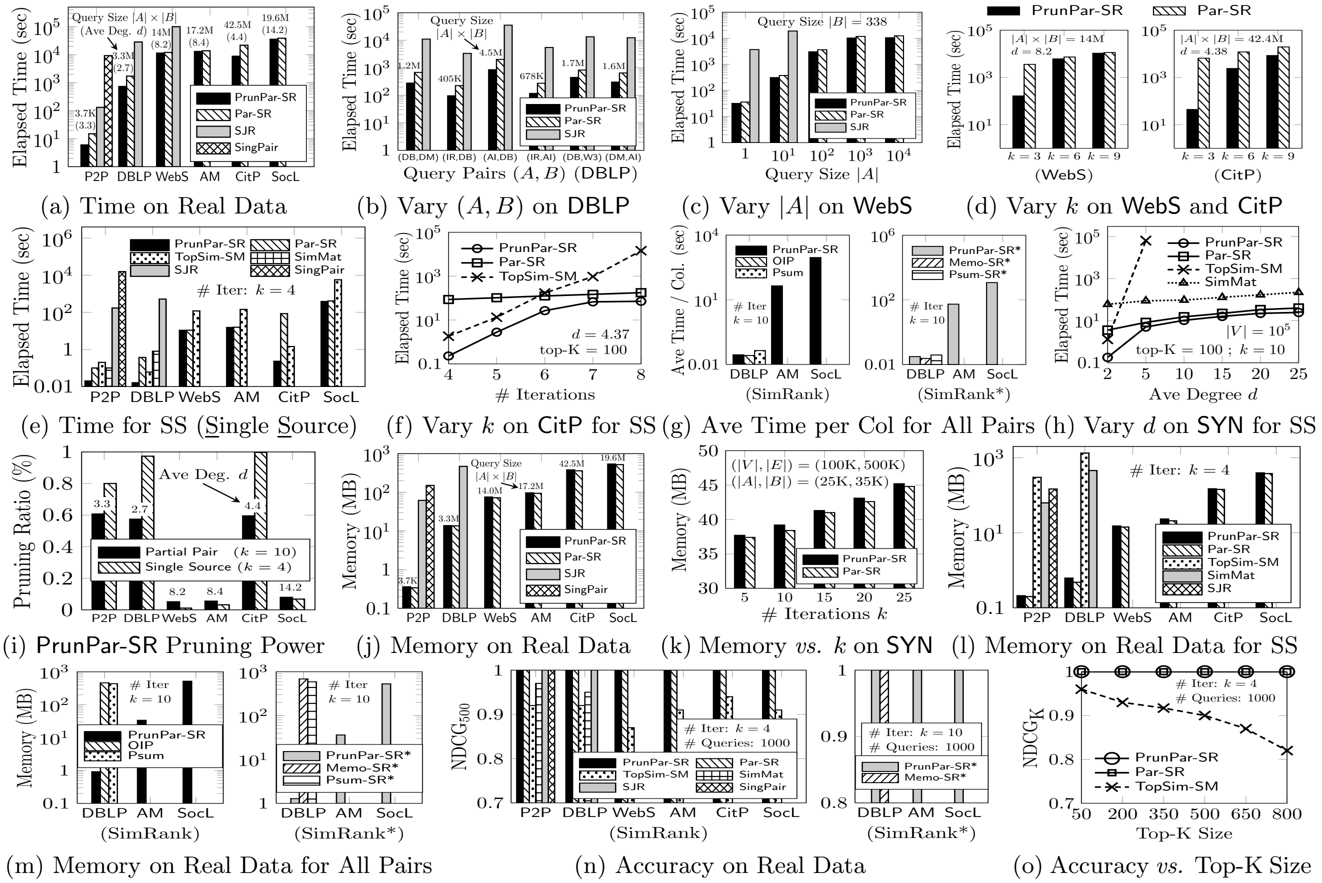
Figure 6c indicates how elapsed time changes with $|A|$ on WebS, fixing $|B| = 338$. The trend shows that (1) when $|A|$ grows, the time for PrunPar-SR, Par-SR grows accordingly; however, the growing ratio begins to reduce when $|A| > 10^3$, which is due to the "right-to-left association" trick in our "seed germination" model that swaps A and B if $|A| > |B|$. (2) SJR crashes as $|A| \geq 100$, whereas PrunPar-SR, Par-SR scale well with $|A|$. (3) The speedup of PrunPar-SR on DBLP is more obvious than that on WebS, due to small density.

Figure 6d depicts the effect of iteration k on the PrunPar-SR and Par-SR time on WebS and CitP. We see that (1) the PrunPar-SR and Par-SR time is directly proportional to k. (2) When $k = 6, 9$, the time difference between Par-SR and PrunPar-SR on CitP is more pronounced than that on WebS. This is because CitP has small average degree, compared

[8] http://snap.stanford.edu/data/index.html

[9] http://community.livejournal.com/tartu/profile

[10] http://www.cse.psu.edu/~madduri/software/GTgraph/index.html



(a) Time on Real Data (b) Vary (A, B) on DBLP (c) Vary $|A|$ on WebS (d) Vary k on WebS and CitP

(e) Time for SS (Single Source) (f) Vary k on CitP for SS (g) Ave Time per Col for All Pairs (h) Vary d on SYN for SS

(i) PrunPar-SR Pruning Power (j) Memory on Real Data (k) Memory *vs.* k on SYN (l) Memory on Real Data for SS

(m) Memory on Real Data for All Pairs (n) Accuracy on Real Data (o) Accuracy *vs.* Top-K Size

Figure 6: Performance Evaluations on Real and Synthetic Datasets

with WebS, which agrees with our complexity predictions that d has a large impact on the PrunPar-SR.

Figure 6e compares the time efficiency of the single-source SimRank algorithms on six real datasets when $k = 4$. The results show that (1) The PrunPar-SR and Par-SR time exhibit a similar trend to partial-pairs assessment in Figure 6a. (2) The speedup of PrunPar-SR is more obvious on P2P, DBLP, CitP due to their small average degree. (3) PrunPar-SR consistently outperforms TopSim-SM by +9.2x; however, on DBLP and CitP, TopSim-SM is faster than Par-SR. This is due to the small degrees of DBLP and CitP, implying that TopSim-SM is more sensitive to d than Par-SR, given k.

Figure 6f presents how the time for PrunPar-SR, Par-SR, TopSim-SM changes with k on CitP for single-source queries. (1) When $k < 6$, TopSim-SM is faster than Par-SR; however, when $k > 6$, Par-SR outperforms TopSim-SM. (2) When k grows, the linear increasing trend of TopSim-SM in the log-y scale axis indicates that the TopSim-SM time will *exponentially* increase with respect to k with fixing degree d, which highlights a limitation of TopSim-SM for computing scores of *all* nodes *w.r.t.* a query. In contrast, the PrunPar-SR time increases with $k < 6$, but this increase slows greatly when $k > 6$, due to its pruning on "seed germination" model.

Figure 6g compares the average time to compute each column of all-pairs SimRank matrix for PrunPar-SR, OIP and Psum, and the all-pairs SimRank* for PrunPar-SR*, Memo-SR* and Psum-SR*, respectively. For SimRank, OIP and Psum crash on AM and SocL due to their large memory for storing all-pairs scores from the previous iteration. For SimRank*, this also occurs on Memo-SR* and Psum-SR* for the same reason. However, PrunPar-SR* can execute on large graphs, due to our partial-pairs method that computes the entire SimRank matrix column by column independently, where each column computation can fit into memory.

Figure 6h depicts the effect of d on the time for PrunPar-SR, Par-SR, TopSim-SM, SimMat on a synthetic dataset for single-source SimRank assessment with $k = 10$ and $|V| =$ 100K. We can see that (1) when d increases, TopSim-SM crashes for $d \geq 6$ due to the TopSim-SM time being too sensitive to d when k is large; however, for the remaining algorithms, the increasing trend is relatively steep *w.r.t.* d. (2) when d is smaller (< 5), PrunPar-SR is much better than Par-SR, illustrating the effectiveness of our pruning technique on graphs with low density. (3) SimMat is less sensitive to d, due to its SVD that may destroy graph sparsity.

Figure 6i reports the pruning power of PrunPar-SR on six real datasets for partial-pair and single-source SimRank assessment, respectively. The pruning ratio is defined as

$$(1 - \frac{\text{\# of edges accessed by PrunPar-SR}}{\text{\# of edges accessed by Par-SR}}) \times 100\%.$$

The results show that the pruning power is more significant on graphs with small degrees (P2P, DBLP, CitP), but is less efficient on graphs with larger degrees (WebS, AM). The larger the degree d, the fewer the number of nonzeros in $\mathbf{u}_k$ or $\mathbf{U}_k$ during our "seed germination" iteration, as expected.

5.2.2 Memory Efficiency

Figure 6j shows the memory efficiency of PrunPar-SR, Par-SR, SJR, SingPair for partial-pairs assessment on real data. We observe that SJR requires more memory than PrunPar-SR and Par-SR on P2P, DBLP; and it crashes on the remaining four datasets. This is because SJR needs to store the h-go cover set that could be very large on the tensor graph, whereas PrunPar-SR and Par-SR require to memoize only auxiliary $\mathbf{U}_k$ for k iterations.

Figure 6k shows the memory of PrunPar-SR and Par-SR *w.r.t.* the growing k on synthetic data, when query size of (A, B) is fixed. It can be discerned that (1) when k grows, the memory consumption increases steadily. This is because PrunPar-SR and Par-SR require to memoize k intermediate $\mathbf{U}_k$ after k iterations. (2) Given k, PrunPar-SR needs slightly more memory than Par-SR, due to its hashing strategy.

Figure 6l compares the memory for single-source SimRank on real data. For $k = 4$, TopSim-SM that computes *all* nodes *w.r.t.* a query only survives on small P2P and DBLP, which requires large memory because SimMap(x) needs to be stored for each node x. SimMat only survives on small P2P and DBLP as well since it requires considerable memory to store the decomposed matrices vis SVD. SJR crashes on all the datasets except for P2P because it needs to find an h-go cover set on a large tensor graph. In contrast, PrunPar-SR and Par-SR for single source SimRank are highly memory efficient since they only need to store k vectors.

Figure 6m shows the memory of the all-pairs SimRank and SimRank* algorithms on DBLP, AM, SocL. On AM and SocL, OIP and Psum crashes for SimRank, whereas Memo-SR* and Psum-SR* crash for SimRank* because they require quadratic memory space to store all-pairs similarities from the previous iterations. In contrast, PrunPar-SR and PrunPar-SR* compute the whole similarity matrix column by column, requiring considerably less memory.

5.2.3 Accuracy

Figure 6n shows the accuracy of single-source algorithms for both SimRank and SimRank* on real datasets. We randomly select 1000 queries, use the average NDCG_{500} as accuracy measure. Using the all-pairs similarities as the baselines, our results on all the datasets show that for SimRank (*resp.* SimRank*), PrunPar-SR, Par-SR, SJR, SingPair (*resp.* PrunPar-SR*, Memo-SR*) do not sacrifice accuracy for high computational efficiency. However, the accuracy for TopSim-SM and SimMat is slightly lower as their computational paradigms are based on top-K search.

Finally, Figure 6o shows the top-K size affects the $\mathrm{NDCG}_{\mathrm{K}}$ of PrunPar-SR, Par-SR, TopSim-SM. The results show that with increasing size of top-K, the accuracy of TopSim-SM is gradually reduced, compared with PrunPar-SR and Par-SR which do not compromise accuracy at all, as expected.

6. CONCLUSIONS

This paper focuses on efficient computation of partial-pairs SimRank. (1) A "seed germination" model is proposed to compute partial-pairs SimRank in $O(k|E|\min\{|A|, |B|\})$ time and $O(|E| + k|V|)$ memory. (2) A pruning strategy is devised to skip redundant edges access for further speeding up partial-pairs computation to $O(m\min\{|A|, |B|\})$ time with $m \leq \min\{k|E|, \Delta^{2k}\}$. As a by-product, our partial-pairs SimRank model not only as a special case improves the computation of the fastest known single-source SimRank, but also induces a memory-efficient algorithm for all-pairs SimRank that can break "high iteration coupling". Our techniques can be readily extended to partial-pairs SimRank* computation. Finally, our experimental results on real and synthetic data have verified the superiority of our algorithms on large graphs against the baselines.

Acknowledgment

This work forms part of the Big Data Technology for Smart Water Network research project funded by NEC Corporation, Japan.

7. REFERENCES

[1] I. Antonellis, H. G. Molina, and C. Chang. SimRank++: Query rewriting through link analysis of the click graph. *PVLDB*, 1(1):408–421, 2008.

[2] P. Berkhin. Survey: A survey on PageRank computing. *Internet Mathematics*, 2(1):73–120, 2005.

[3] D. Fogaras and B. Rácz. Scaling link-based similarity search. In *WWW*, pages 641–650, 2005.

[4] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for SimRank. In *ICDE*, pages 589–600, 2013.

[5] G. He, H. Feng, C. Li, and H. Chen. Parallel SimRank computation on large graphs with iterative aggregation. In *KDD*, pages 543–552, 2010.

[6] J. He, H. Liu, J. X. Yu, P. Li, W. He, and X. Du. Assessing single-pair similarity over graphs by aggregating first-meeting probabilities. *Inf. Syst.*, 42:107–122, 2014.

[7] G. Jeh and J. Widom. SimRank: A measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

[8] R. Jin, V. E. Lee, and H. Hong. Axiomatic ranking of network role similarity. In *KDD*, pages 922–930, 2011.

[9] M. Kusumoto, T. Maehara, and K. ichi Kawarabayashi. Scalable similarity search for SimRank. In *SIGMOD Conference*, pages 325–336, 2014.

[10] P. Lee, L. V. S. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In *ICDE*, pages 774–785, 2012.

[11] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of SimRank for static and dynamic information networks. In *EDBT*, pages 465–476, 2010.

[12] P. Li, H. Liu, J. X. Yu, J. He, and X. Du. Fast single-pair SimRank computation. In *SDM*, pages 571–582, 2010.

[13] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for SimRank computation. *VLDB J.*, 19(1):45–66, 2010.

[14] L. Sun, R. Cheng, X. Li, D. W. Cheung, and J. Han. On link-based similarity join. *PVLDB*, 4(11):714–725, 2011.

[15] W. Yu, X. Lin, and J. Le. Taming computational complexity: Efficient and parallel SimRank optimizations on undirected graphs. In *WAIM*, pages 280–296, 2010.

[16] W. Yu, X. Lin, and W. Zhang. Towards efficient SimRank computation on large networks. In *ICDE*, pages 601–612, 2013.

[17] W. Yu, X. Lin, and W. Zhang. Fast incremental SimRank on link-evolving graphs. In *ICDE*, pages 304–315, 2014.

[18] W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *PVLDB*, 7:13–24, 2013.

[19] W. Yu and J. A. McCann. Sig-SR: SimRank search over singular graphs. In *SIGIR*, pages 859–862, 2014.

[20] W. Zheng, L. Zou, Y. Feng, L. Chen, and D. Zhao. Efficient SimRank-based similarity join over large graphs. *PVLDB*, 6(7):493–504, 2013.