Fast All-Pairs SimRank Assessment on Large Graphs and Bipartite Domains

Weiren Yu, *Member, IEEE*, Xuemin Lin, Wenjie Zhang, and Julie A. McCann, *Member, IEEE*

Abstract—SimRank is a powerful model for assessing vertex-pair similarities in a graph. It follows the concept that two vertices are similar if they are referenced by similar vertices. The prior work [18] exploits partial sums memoization to compute SimRank in $O(Kmn)$ time on a graph of n vertices and m edges, for K iterations. However, computations among different partial sums may have redundancy. Besides, to guarantee a given accuracy ϵ, the existing SimRank needs $K = \lceil \log_C \epsilon \rceil$ iterations, where C is a damping factor, but the geometric rate of convergence is slow if a high accuracy is expected. In this paper, (1) a novel clustering strategy is proposed to eliminate duplicate computations occurring in partial sums, and an efficient algorithm is then devised to accelerate SimRank computation to $O(Kd'n^2)$ time, where d' is typically much smaller than $\frac{m}{n}$. (2) A new differential SimRank equation is proposed, which can represent the SimRank matrix as an exponential sum of transition matrices, as opposed to the geometric sum of the conventional counterpart. This leads to a further speedup in the convergence rate of SimRank iterations. (3) In bipartite domains, a novel finer-grained partial max clustering method is developed to speed up the computation of the 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 can be typically much smaller than m. 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 5X-12X faster than the baselines.

Index Terms—Structural similarity, SimRank, hyperlink analysis

1 Introduction

Identifying similar objects based on link structure is a fundamental operation for many web mining tasks. Examples include webpage ranking [3], hypertext classification (KNN) [14], graph clustering (K-means) [4], and collaborative filtering [12]. 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 percent of new objects are created weekly [5]. 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.

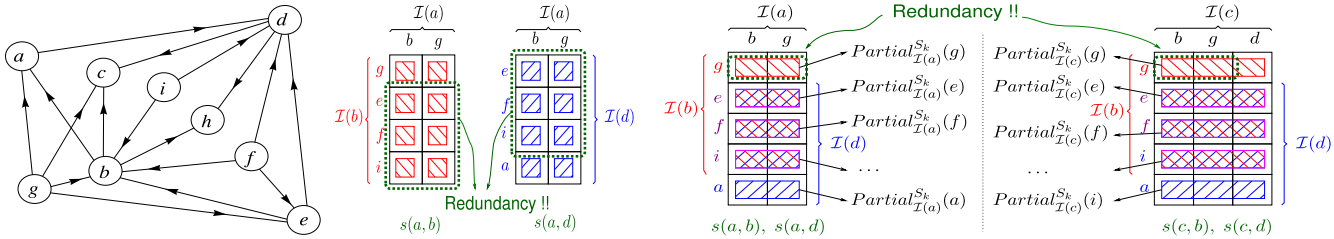
Amid the existing similarity metrics, SimRank [12] has emerged as a powerful tool for assessing structural similarities between two objects. Similar to the well-known PageRank [3], SimRank scores depend merely on the web link structure, independent of the textual content of objects. The major difference 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 [12], and has gained increasing popularity in many areas such as bibliometrics [15], top-K search [14], and recommender systems [1]. 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" [12]. 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.

Among the existing SimRank computing problems, *all-pairs* SimRank assessment (i.e., finding similarities for all pairs of vertices) is more important than *single-source* SimRank assessment (i.e., finding similarities between a query vertex and all other vertices) since, in many real applications, people are often interested in not only node ranking (e.g., "Which objects are similar to a certain query object?"), but also node-pair ranking (e.g., "What are the top-K most similar pairs of objects in a graph?"). Generally, all-pairs SimRank contains similarity information that can handle both node and node-pair ranking problems. The best known algorithm for computing all-pairs SimRank was proposed by Lizorkin et al. [18] (hereafter referred to as psum-SR), which 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 [18] resides in 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

- *W. Yu and J.A. McCann are with the Department of Computing, Imperial College London, United Kingdom. E-mail: {weiren.yu, jamm}@imperial.ac.uk.*
- *X. Lin and W. Zhang are with the School of Computer Science and Engineering, University of New South Wales, Australia. E-mail: {lxue, zhangw}@cse.unsw.edu.au.*

Manuscript received 5 Nov. 2013; revised 18 Mar. 2014; accepted 8 Apr. 2014. Date of publication 15 July 2014; date of current version 1 June 2015. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TKDE.2014.2339828



(a) A paper citation network (b) Naive method in [12] (c) psum-SR leads to redundancy among different partial sums

Fig.1 Merit and demerit of *partial sums memoizing* for SimRank computation on a paper citation network.

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 $O(Kd^2n^2)$ to $O(Kdn^2)$,[1] where d is the average in-degree in a graph.

Before shedding light on the blemish of psum-SR [18], let us first revisit the central idea of partial sums memoizing, as depicted in the following example:

Example 1. Consider a paper citation network $\mathcal{G}$ in Fig. 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 Fig. 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 Fig. 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 Fig. 1c).

Despite the aforementioned merits of psum-SR, existing work [18] on SimRank has some limitations.

First, we observe from Example 1 that computing partial sums over different in-neighbor sets may have redundancy. For instance, $\mathcal{I}(a)$ and $\mathcal{I}(c)$ in Fig. 1c have two vertices $\{b,g\}$ in common, implying that the sub-summation $Partial^{s_k}_{\{b,g\}}(\bigstar)$ is the common part shared between the partial sums $Partial^{s_k}_{\mathcal{I}(a)}(\bigstar)$ and $Partial^{s_k}_{\mathcal{I}(c)}(\bigstar)$. Thus, there is an opportunity to speed up the computation of SimRank by preprocessing the common sub-summation $Partial^{s_k}_{\{b,g\}}(\bigstar)$ once, and caching it for both $Partial^{s_k}_{\mathcal{I}(a)}(\bigstar)$ and $Partial^{s_k}_{\mathcal{I}(c)}(\bigstar)$ 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.

Second, the existing iterative paradigm [18] 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 [13]. For instance, our experiments on a DBLP citation network shows that a desired accuracy of $\epsilon = 0.001$ may lead to more than 30 iterations of SimRank, for the damping factor $C = 0.8$. Lizorkin et al. has proved theoretically in [18] 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 [12, Section 4.3.2], called the minimax variation SimRank, may also have duplicate 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 may be 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 5.

Contributions. Below are our main contributions:

- We propose an adaptive clustering strategy based on a minimum spanning tree to eliminate duplicate computations in partial sums [18] (Section 3). By optimizing the sub-summations sharing among different partial sums, an efficient algorithm is devised for speeding up the computation of SimRank from $O(Kdn^2)$ [18] 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 4). We show that the new notion of SimRank can be characterized

1. As $n \cdot d = m$, $O(Kmn)$ time in [18] is equivalent to $O(Kdn^2)$.

2. Recall from [18] that *a partial sum* for a binary function $f : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ over a set $\mathcal{D} = \{x_1, \ldots, x_n\} \subseteq \mathcal{X}$, denoted by $Partial^f_{\mathcal{D}}(\bigstar)$, is defined as

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

as an exponential sum in terms of the transition matrix while fairly preserving the relative order of SimRank. We also devise a space-efficient iterative paradigm for computing differential SimRank, which integrates 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 [12, 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 5).
- We conduct extensive experiments on real and synthetic data sets (Section 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; (3) for the Minimax SimRank variation in bipartite domains, our finer-grained partial max sharing method is 5X-12X faster than the baselines in CPU time.

Related work. The earliest mention of SimRank dates back to Jeh and Widom [12] 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. [18] 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 [18] in the following. (i) We put forward the phenomenon of partial sums redundancy in [18] that typically exists in real graphs. (ii) We accelerate the convergence of SimRank iterations from geometric [18] to exponential growth. (iii) In bipartite domains, we also develop techniques of partial max sharing for the Minimax SimRank variation model.

There has also been a flurry of interests (e.g., [1], [6], [11], [14], [15], [16]) in SimRank optimization. Li et al. [15] 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 [15] 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. [11] 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 [1] to address the issue of query rewriting for sponsored search. Fogaras and Rácz [6] adopted a scalable Monte Carlo sampling approach to estimate SimRank by using the first meeting time of two random surfers. Li et al. [16] employed an effective method for locally computing single-pair SimRank by breaking the holistic nature of the SimRank recursion. Lee et al. [14] devised a top-K SimRank algorithm needing to access only a small fraction of vertices in a graph. Most recently, Fujiwara et al. [7] proposed an excellent SVD-based SimRank for efficiently finding the top-k similar nodes w.r.t. a given query.

2 Preliminaries

We revisit the two forms of SimRank, i.e., the iterative form [12], [18], and the matrix form [11], [15]. The consistency of two forms was pointed out in [15].

2.1 Iterative Form of SimRank

For a digraph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with a vertex set $\mathcal{V}$ and an edge set $\mathcal{E}$, let $\mathcal{I}(a)$ be the in-neighbor set of a, i.e.,

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

The SimRank score between vertices a and b, denoted by $s(a, b)$, is defined as (i) $s(a, a) = 1$; (ii) $s(a, b) = 0$, if $\mathcal{I}(a) = \varnothing$ or $\mathcal{I}(\mathrm{b}) = \varnothing$; (iii) otherwise,

$$s(a, b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} s(i, j), \tag{1}$$

where $C \in (0, 1)$ is a damping factor, and $|\mathcal{I}(a)|$ is the cardinality of $\mathcal{I}(a)$.

The above formulas naturally lead to the iterative method. Start with $s_0(a, b) = \{^{1,\ a=b;}_{0,\ a \neq b.}$ and for $k = 0, 1, \ldots$, 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,

$$s_{k+1}(a, b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{i \in \mathcal{I}(a)} s_k(i, j). \tag{2}$$

The resultant sequence $\{s_k(a, b)\}_{k=0}^{\infty}$ converges to $s(a, b)$, the *exact* solution of Eq. (1).

2.2 Matrix Form of SimRank

In matrix notations, SimRank can be formulated as

$$\mathbf{S} = C \cdot (\mathbf{Q} \cdot \mathbf{S} \cdot \mathbf{Q}^T) + (1 - C) \cdot \mathbf{I}_n, \tag{3}$$

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.

3 Eliminating Partial Sums Duplicate Computations

The existing method psum-SR [18] of performing Eq. (2) is to memoize the partial sums over $\mathcal{I}(a)$ first:

$$Partial^{s_k}_{\mathcal{I}(a)}(j) = \sum_{i \in \mathcal{I}(a)} s_k(i,j), \quad (j \in \mathcal{I}(b)) \tag{4}$$

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

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} Partial^{s_k}_{\mathcal{I}(a)}(j). \tag{5}$$

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 Fig. 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 show how to optimize sub-summations sharing among different partial sums in this section.

3.1 Partition In-Neighbor Sets for (Inner) Partial Sums Sharing

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

Definition 1. 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 $\mathcal{D}$:*

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

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 Fig. 1a.

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

Proposition 1. *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*

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{j \in \mathcal{I}(b)} \sum_{\Delta \in \mathscr{P}(\mathcal{I}(a))} Partial^{s_k}_{\Delta}(j). \tag{6}$$

Here, $Partial^{s_k}_{\Delta}(j)$ is defined as Eq. (4) with $\mathcal{I}(a)$ replaced by Δ.

Sketch of Proof. The proof follows immediately from the facts that (i) for two disjoint sets $\mathcal{A}$ and $\mathcal{B}$, $Partial^{s_k}_{\mathcal{A}}(j) + Partial^{s_k}_{\mathcal{B}}(j) = Partial^{s_k}_{\mathcal{A}\cup\mathcal{B}}(j)$, $\forall j$, and (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 pre-computing 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 Fig. 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, psum-SR [18] has to start from scratch to compute $Partial^{s_k}_{\mathcal{I}(a)}(\star)$ and $Partial^{s_k}_{\mathcal{I}(c)}(\star)$, independently, 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. (6) can be simplified to the conventional psum-SR iteration in Eq. (5). 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 $\Delta \in \mathscr{P}(\mathcal{I}(a))$ is 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 similarity values to be clustered. An extreme example is 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. (6) would deteriorate to the naive iteration [12] in Eq. (2), 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* (OIP), can be formulated as follows:

Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, OIP is to find the optimal partition $\mathscr{P}(\mathcal{I}(a)) = \{\Delta^i_a \mid i = 1, \ldots, |\mathscr{P}(\mathcal{I}(a))|\}$ 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 $\mathcal{I}(a)$, $a \in \mathcal{V}$, is minimized by reusing the sub-summation results $Partial^{s_k}_{\Delta^i_a}(\star)$ over chunks Δ^i_a.

Proposition 2. *The OIP problem is NP-hard.*

(Please refer to Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TKDE.2014.2339828, for a detailed proof.)

We next seek for a good heuristic method for OIP.

Main Idea. 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]

$$\mathcal{TC}_{\mathcal{I}(a)\to\mathcal{I}(b)} \triangleq \min\{|\mathcal{I}(a) \ominus \mathcal{I}(b)|, |\mathcal{I}(b)| - 1\}, \tag{7}$$

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

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,

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

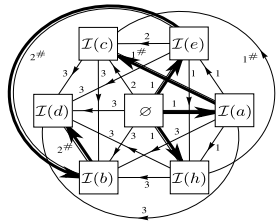
As an example in Fig 1c, given $\mathcal{I}(b) = \{g,e,f,i\}$ and $\mathcal{I}(d) = \{e,f,i,a\}$, we have $\mathcal{I}(b) \ominus \mathcal{I}(d) = \{g,a\}$.

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

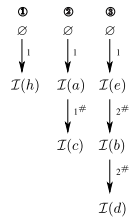
(a) In-neighbors in $\mathcal{G}$

	$\mathcal{I}(a)$	$\mathcal{I}(e)$	$\mathcal{I}(h)$	$\mathcal{I}(c)$	$\mathcal{I}(b)$	$\mathcal{I}(d)$
$\varnothing$	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 $\mathscr{T}$ of $\mathscr{G}$



(d) Partial Sums Order

Fig 2. Constructing a minimum spanning tree $\mathscr{T}$ to find an optimized topological sort for partial sums sharing.

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 $\mathcal{TC}_{\mathcal{I}(a)\rightarrow\mathcal{I}(b)}$. 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) $Partial^{s_k}_{\mathcal{I}(a)}(\bigstar)$ should be computed prior to $Partial^{s_k}_{\mathcal{I}(b)}(\bigstar)$ 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 $Partial^{s_k}_{\mathcal{I}(a)}(\bigstar)$ can be cached and shared with $Partial^{s_k}_{\mathcal{I}(b)}(\bigstar)$ computation.

The following example depicts how this idea works:

Example 2. Consider the network $\mathcal{G}$ in Fig. 1a, with the vertices and the corresponding non-empty in-neighbor sets depicted in Fig. 2a. We show how to find a decent ordering for partial sums computing and sharing in $\mathcal{G}$.

First, we compute the transition cost of each pair of in-neighbor sets (along with an empty set $\varnothing$) in $\mathcal{G}$, by using Eq. (7). The results are shown in Fig. 2b, where each cell describes the transition cost from the in-neighbor set in the left most column 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 $\mathcal{TC}_{\mathcal{I}(e)\rightarrow\mathcal{I}(b)} = 2$. This cell is tagged with #, indicating that the partial sum $Partial^{s_k}_{\mathcal{I}(b)}(\bigstar)$ can be computed from the memoized result of $Partial^{s_k}_{\mathcal{I}(e)}(\bigstar)$ (rather than from scratch). This is because the transition cost 2 is, in essence, obtained from the two 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. (7). Note that the lower triangular part of the table in Fig. 2b remains empty since we are interested only in the cost $\mathcal{TC}_{\mathcal{I}(x)\rightarrow\mathcal{I}(y)}$ when $|\mathcal{I}(x)| \leq |\mathcal{I}(y)|$.

Next, we build a weighted digraph $\mathscr{G}$ in Fig. 2c, with vertices corresponding to the non-empty in-neighbor sets (plus $\varnothing$) of $\mathcal{G}$ (which are in column '$\mathcal{I}(\bigstar)$' of Fig. 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 Fig. 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) $Partial^{s_k}_{\mathcal{I}(e)}(\bigstar)$ is computed from scratch (from $\varnothing$) with one operation, and (ii) $Partial^{s_k}_{\mathcal{I}(b)}(\bigstar)$ is obtained by reusing the result of $Partial^{s_k}_{\mathcal{I}(e)}(\bigstar)$, involving two 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 Fig. 2c. Consequently, using depth-first search (DFS), we can obtain 3 paths from $\mathscr{T}$ for partial sums optimization, as shown in Fig. 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 3 shows the correctness.

Proposition 3. *Given two distinct non-empty in-neighbor sets $\mathcal{I}(a)$ and $\mathcal{I}(b)$, and a partial sum $Partial^{s_k}_{\mathcal{I}(a)}(\bigstar)$, 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

$$\mathcal{I}(b) = (\mathcal{I}(b) \cap \mathcal{I}(a)) \cup (\mathcal{I}(b) \backslash \mathcal{I}(a)). \tag{8}$$

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

$$\begin{aligned} Partial^{s_k}_{\mathcal{I}(b)}(y) = Partial^{s_k}_{\mathcal{I}(a)}(y) &- \sum_{x \in \mathcal{I}(a) \backslash \mathcal{I}(b)} s_k(x, y) \\ &+ \sum_{x \in \mathcal{I}(b) \backslash \mathcal{I}(a)} s_k(x, y), \quad (y \in \mathcal{V}) \end{aligned} \tag{9}$$

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

Sketch of Proof. The proof of Eq. (8) is trivial, whereas the proof of Eq. (9) is based on the facts that (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 Appendix B, available in the online supplemental material, we give an illustrative example to show how to find all the partitions of in-neighbor sets for partial sums sharing via Proposition 3.

3.2 Use In-Neighbor Set Partitions for Outer Sums Sharing

After in-neighbor set partitions have been identified for *(inner)* partial sums sharing, optimization methods in this section allow *outer* partial sums sharing for further speeding up SimRank computation.

To avoid ambiguity, we refer to the sums w.r.t. the index i in Eq. (4) as *(inner) partial sums*, and the sums w.r.t. the index j in Eq. (5) as *outer partial sums*.

Our key observation is as follows. Recall from Eq. (5) 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 Fig. 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 $s(a,d)$ 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 Fig. 1c, for computing $s(c,b)$ and $s(c,d)$, the sum of the three horizontal bars at rows 'e', 'f', 'i' is again a repeated operation. As such, the major problem of Eq. (5) 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. (5) is to split $\mathcal{I}(b)$ into several chunks Δ^i_b first, such that

$$\mathscr{P}(\mathcal{I}(b)) = \{\Delta^i_b \mid i = 1, \ldots, |\mathscr{P}(\mathcal{I}(b))|\},$$

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

$$s_{k+1}(a,b) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(b)|} \sum_{\Delta^i_b \in \mathscr{P}(\mathcal{I}(b))} OuterPartial^{\mathcal{I}(a),s_k}_{\Delta^i_b} \qquad (10)$$

with

$$OuterPartial^{\mathcal{I}(a),s_k}_{\Delta^i_b} \triangleq \sum_{j \in \Delta^i_b} Partial^{s_k}_{\mathcal{I}(a)}(j).$$

In contrast with Eq. (5), our method in Eq. (10) can eliminate the redundancy among different outer partial sums. Once computed, the outer partial sum $OuterPartial^{\mathcal{I}(a),s_k}_{\Delta^i_b}$ is memoized and can be reused later without recalculation again. As an example in Fig. 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 $OuterPartial^{\mathcal{I}(a),s_k}_{\{e,f,i\}}$ 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. Thus, the partitioning techniques for (inner) partial sums sharing in Section 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. (8) for (inner) partial sums sharing, once identified, can be reused later for outer partial sums sharing. The correctness is verified in Proposition 4.

Proposition 4. *Given two non-empty in-neighbor sets $\mathcal{I}(b)$ and $\mathcal{I}(d)$, an outer partial sum $OuterPartial^{\mathcal{I}(a),s_k}_{\mathcal{I}(b)}$, and (inner) partial sums $Partial^{s_k}_{\mathcal{I}(a)}(\star)$, if $|\mathcal{I}(b) \ominus \mathcal{I}(d)| < |\mathcal{I}(d)| - 1$, then we have the following:*

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

$$s_{k+1}(a,d) = \frac{C}{|\mathcal{I}(a)||\mathcal{I}(d)|} \text{ then } OuterPartial^{\mathcal{I}(a),s_k}_{\mathcal{I}(d)}, \quad \forall a \in \mathcal{V} \setminus \{d\}. \qquad (11)$$

(The proof is similar to Proposition 3. We omit it here.)

In Appendix B, available in the online supplemental material, we provide an example to illustrate how to use outer partial sums sharing for further speeding up the computation of SimRank.

3.3 An Algorithm for Computing SimRank

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 section is the following:

Proposition 5. *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 5 by providing an algorithm for SimRank computation, with the desired complexity bound.

Algorithm. The algorithm, referred to as OIP-SR, is shown in Algorithm 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.

(Please refer to Appendix C, available in the online supplemental material, for the detailed descriptions of algorithm OIP-SR and procedures OP and DMST-Reduce.)

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

Algorithm 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 $k \leftarrow 0, 1, \ldots, K-1$ **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**
        $Partial^{s_k}_{\mathcal{I}(v)}(y) \leftarrow Partial^{s_k}_{\mathcal{I}(u)}(y) - \sum_{x \in \mathcal{I}(u) \setminus \mathcal{I}(v)} s_k(x, y) + \sum_{x \in \mathcal{I}(v) \setminus \mathcal{I}(u)} s_k(x, y)$;
      $s_{k+1}(v, \star) \leftarrow$ 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 $s_K(\star, \star)$;

(Please see Appendix D, available in the online supplemental material, for the detailed analysis.)

4 Exponential Rate of Convergence for SimRank Iterations

For a desired accuracy ϵ, the existing paradigm (via Eq. (2)) for computing SimRank needs $K = \lceil \log_C \epsilon \rceil$ iterations [18]. 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. (3))

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

as a power series:

$$\mathbf{S} = (1 - C) \cdot \sum_{i=0}^{\infty} C^i \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i, \tag{12}$$

where we notice that the coefficient for each term in the summation makes a geometric sequence $\{1, C, C^2, \ldots\}$. 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. (12) as follows:

$$\hat{\mathbf{S}} = e^{-C} \cdot \sum_{i=0}^{\infty} \frac{C^i}{i!} \cdot \mathbf{Q}^i \cdot (\mathbf{Q}^T)^i, \tag{13}$$

Comparing Eq. (12) with Eq. (13), 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. (13) that yields the exponential sequence $\{1, \frac{C}{1!}, \frac{C^2}{2!}, \ldots\}$ 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.

4.1 Closed Form of Exponential SimRank

With the modified notion of SimRank in Eq. (13), we now need to define an Eq. (3)-like recurrence for $\hat{\mathbf{S}}$.

Definition 2. *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:*

$$\frac{d\hat{\mathbf{S}}(t)}{dt} = \mathbf{Q} \cdot \hat{\mathbf{S}}(t) \cdot \mathbf{Q}^T, \qquad \hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n. \tag{14}$$

Note that the solution of Eq. (14) is unique since the initial condition $\hat{\mathbf{S}}(0) = e^{-C} \cdot \mathbf{I}_n$ is specified. Based on Definition 2, it is crucial to verify that $\hat{\mathbf{S}}$ (in Eq. (13)) is the solution to Eq. (14). Proposition 6 shows the correctness.

Proposition 6. *The matrix differential form of SimRank in Eq. (14) has an exact solution $\hat{\mathbf{S}}$ given in Eq. (13).*

(Please refer to Appendix A, available in the online supplemental material, for a detailed proof.)

To iteratively compute $\hat{\mathbf{S}}$, the conventional way is to use *the Euler method* [2] 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. (14) 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

$$\begin{cases} \mathbf{T}_{k+1} = \mathbf{Q} \cdot \mathbf{T}_k \cdot \mathbf{Q}^T \\ \hat{\mathbf{S}}_{k+1} = \hat{\mathbf{S}}_k + e^{-C} \cdot \frac{C^{k+1}}{(k+1)!} \cdot \mathbf{T}_{k+1} \end{cases} \text{with} \begin{cases} \mathbf{T}_0 = \mathbf{I}_n \\ \hat{\mathbf{S}}_0 = e^{-C} \cdot \mathbf{I}_n. \end{cases} \tag{15}$$

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. (15) equals the sum of the first k terms of the infinite series $\hat{\mathbf{S}}$ in Eq. (13).

4.2 A Space-Efficient Iterative Paradigm

Although the paradigm of Eq. (15) can iteratively compute $\hat{\mathbf{S}}_k$ that converges to the exponential SimRank $\hat{\mathbf{S}}$, we observe that Eq. (15) requires additional memory space to store the intermediate result $\mathbf{T}_k$ per iteration. In this section, we provide an improved version of Eq. (15) that can produce the same result without using extra space for caching $\mathbf{T}_k$.

Proposition 7. *Given any total iteration number K, the following paradigm can be used to iteratively compute $\tilde{\mathbf{S}}_K$:*

$$\begin{cases} \tilde{\mathbf{S}}_0 = e^{-C} \cdot \mathbf{I}_n, \\ \tilde{\mathbf{S}}_{k+1} = \frac{C}{K-k} \cdot \mathbf{Q} \cdot \tilde{\mathbf{S}}_k \cdot \mathbf{Q}^T + e^{-C} \cdot \mathbf{I}_n. \quad (k = 0, \ldots, K-1). \end{cases} \tag{16}$$

The result of $\tilde{\mathbf{S}}_K$ at the last iteration is exactly the same as $\hat{\mathbf{S}}_K$ in Eq. (15).

The main idea of our improved paradigm Eq. (16) is based on two observations: (1) For every iteration $k = 0, 1, \ldots, K$, the result of $\hat{\mathbf{S}}_k$ in Eq. (15) is actually the sum of the first k terms of the infinite series $\hat{\mathbf{S}}$ in Eq. (13). (2) For any total iteration number K, the result of $\tilde{\mathbf{S}}_K$ at the last iteration in Eq. (16) equals the sum of the first K terms of the infinite series $\hat{\mathbf{S}}$ in Eq. (13). Both of these observations can be readily verified by direct inductive manipulations. As an example for $K = 3$, our improved paradigm Eq. (16) 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. (16) over Eq. (15) is the space efficiency—in Eq. (16), we do not need to use an auxiliary matrix $\mathbf{T}_k$ to store the temporary results. Moreover, since Eq. (16) has a very similar form to the SimRank matrix form in Eq. (3), our partial sums sharing techniques in Section 3 can be directly applied to the iterative form of Eq. (16), i.e., when $a \neq b$, for $k = 0, 1, \ldots, K-1$,

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

It is worth noticing that in Eq. (15), we can iteratively compute $\hat{\mathbf{S}}_{k+1}$ from $\hat{\mathbf{S}}_k$ for *any* $k = 0, 1, \ldots,$ whereas, in Eq. (16), for any given K, we can only iteratively compute $\tilde{\mathbf{S}}_{k+1}$ from $\tilde{\mathbf{S}}_k$ for $k = 0, 1, \ldots, K-1$, 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. (16), in contrast with Eq. (15) in which K can be determined in an either a-priori or a-posteriori style. Fortunately, this requirement is not an obstacle to Eq. (16), since in the next section we will show a nice a-priori bound of the total iteration number K for Eq. (16) to attain a given accuracy ϵ.

4.3 Error Estimate

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

Proposition 8. *For each iteration $k = 0, 1, 2, \ldots,$ the difference between the kth iterative and the exact similarity matrix in Eqs. (13) and (15) can be bounded as follows:*

$$\|\hat{\mathbf{S}}_k - \hat{\mathbf{S}}\|_{\max} \leq \frac{C^{k+1}}{(k+1)!}, \tag{17}$$

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

(Please refer to Appendix A, available in the online supplemental material, for a detailed proof.)

For the SimRank differential model Eq. (13), Proposition 8 allows finding out the exact number of iterations needed for attaining a desired accuracy, based on the following corollary.

Corollary 1. *For a desired accuracy $\epsilon > 0$, the number of iterations required to perform Eq. (15) is*

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

Here, $W(\star)$ is the Lambert W function [10].

(Please see the Appendix A, available in the online supplemental material, for a detailed proof.)

Noting that $\ln(x) - \ln(\ln(x)) \leq W(x) \leq \ln(x), \forall x > e$ [10], we have the following improved version of Corollary 1, which may avoid computing the Lambert W function.

Corollary 2. *For a desired accuracy $0 < \epsilon < \frac{1}{\sqrt{2\pi}} e^{-C \cdot e^2}$, the number of iterations needed to perform Eq. (15) is*

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

Comparing this with the conventional SimRank model that requires $K = \lceil \log_C \epsilon \rceil$ iterations [18] for a given accuracy ϵ, we see that our revision of the differential SimRank model in Eq. (14) 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 to find out the number of iterations K' in Eq. (15) necessary to our differential SimRank model Eq. (14) as follows:

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

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

In contrast, the conventional SimRank model Eq. (2) needs $K = \lceil \log_{0.8} 0.0001 \rceil = 41$ iterations.

For ranking purpose, our experimental results in Section 6 further show that the revised notion of SimRank in Eq. (14) 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 [18].

5 Partial Max Sharing for Minimax SimRank Variation in Bipartite Graphs

Having investigated the partial *sums* sharing problem for optimizing SimRank computation in Section 4, we now focus on the partial *max* sharing problem for optimizing the computation of *the Minimax SimRank variation*, a model proposed in [12, Section 4.3.2].

Given a bipartite graph $\mathcal{G} = (\mathcal{V} \cup \mathcal{W}, \mathcal{E})$, for any vertex $A \in \mathcal{V}$, the out-neighbor set of A is defined as

$$\mathcal{O}(A) = \{x \in \mathcal{V} \,|\, (A, x) \in \mathcal{E}\}.$$

For every two distinct vertices A and B in $\mathcal{V}$, the similarity of the Minimax SimRank variation, denoted as $s(A, B)$, is defined as follows [12]:

$$s^A(A,B) = \frac{C}{|\mathcal{O}(A)|} \sum_{i \in \mathcal{O}(A)} \max_{j \in \mathcal{O}(B)} s(i,j),$$

$$s^B(A,B) = \frac{C}{|\mathcal{O}(B)|} \sum_{j \in \mathcal{O}(B)} \max_{i \in \mathcal{O}(A)} s(i,j),$$

$$s(A,B) = \min\{s^A(A,B), s^B(A,B)\}.$$

The Minimax SimRank variation model is particularly useful when we sometimes do not need to compare all A's neighbors with all B's. An real application for this model is depicted in Appendix F, available in the online supplemental material.

To compute $s(A, B)$, 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,

$$s^A_{k+1}(A,B) = \frac{C}{|\mathcal{O}(A)|} \sum_{i \in \mathcal{O}(A)} \max_{j \in \mathcal{O}(B)} s_k(i,j), \tag{18}$$

$$s^B_{k+1}(A,B) = \frac{C}{|\mathcal{O}(B)|} \sum_{j \in \mathcal{O}(B)} \max_{i \in \mathcal{O}(A)} s_k(i,j), \tag{19}$$

$$s_{k+1}(A,B) = \min\{s^A_{k+1}(A,B), s^B_{k+1}(A,B)\}. \tag{20}$$

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. (18) [5] as follows:

$$Partial\_Max^{s_k}_{\mathcal{O}(B)}(i) = \max_{j \in \mathcal{O}(B)} s_k(i,j), \tag{21}$$

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

$$s^A_{k+1}(A,B) = \frac{C}{|\mathcal{O}(A)|} \sum_{i \in \mathcal{O}(A)} Partial\_Max^{s_k}_{\mathcal{O}(B)}(i). \tag{22}$$

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. (21) and (22) have a very similar form to Eqs. (4) and (5), we only can apply the (outer) partial sums sharing technique of Section 3.2 to further speed up the summations in Eq. (22), but may not always employ the (inner) partial sums sharing technique of Section 3.1 to accelerate the partial max computation in Eq. (21). The reason is that, for partial *sums* sharing, "subtraction" is allowed to compute one partial sum from another (see Eq. (9) in Proposition 3), 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 3. 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 3, the partial sums $Partial^{s_k}_{\mathcal{O}(D)}(\star)$ can be computed from the memoized $Partial^{s_k}_{\mathcal{O}(B)}(\star)$ as

$$\begin{aligned} Partial^{s_k}_{\mathcal{O}(D)}(\star) = {} & Partial^{s_k}_{\mathcal{O}(B)}(\star) + Partial^{s_k}_{\{g,h,i\}}(\star) \\ & - Partial^{s_k}_{\{c,j\}}(\star). \end{aligned} \tag{23}$$

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. (23)-like approach. This is because "subtraction" is involved in Eq. (23)—although we know

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

we do not know how to derive $Partial\_Max^{s_k}_{\mathcal{O}(D)}(\star)$ from the results of $Partial\_Max^{s_k}_{\mathcal{O}(B)\cup\{g,h,i\}}(\star)$ 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

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

5. In the following, we shall focus solely on optimizing Eq. (18). A similar method can be applied to Eq. (19).

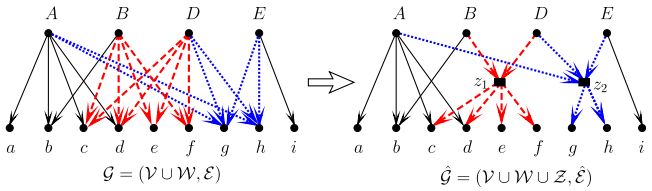


Fig. 3. Edge concentration.

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 $\mathcal{O}(X) \cap \mathcal{O}(Y)$:

$$z(\star) = Partial\_Max^{s_k}_{\mathcal{O}(X)\cap\mathcal{O}(Y)}(\star), \tag{24}$$

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

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

In comparison, the partial sums sharing approach in Section 3, if ported to the partial max sharing, only allows $Partial\_Max^{s_k}_{\mathcal{O}(Y)}(\star)$ being computed from another memoized 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. (24) for all out-neighbor sets sharing, we first introduce the notion of *biclique*.

Definition 3. *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 Fig. 3, $(\{B, D\}, \{c, d, e, f\})$ (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* [17] of a biclique in $\mathcal{G}$. As shown in the right part of Fig. 3, 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 [15]. Thus, to find bicliques in $\mathcal{G}$, we invoke a heuristic [4].

Algorithm. We next present an algorithm for computing Minimax SimRank variation in a bipartite graph. The algorithm, max-MSR, is shown in Appendix E, available in the online supplemental material. 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.

Correctness and complexity. We can verify max-MSR correctly finds $s_K(\star, \star)$, satisfying Eqs. (18)-(20).

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.

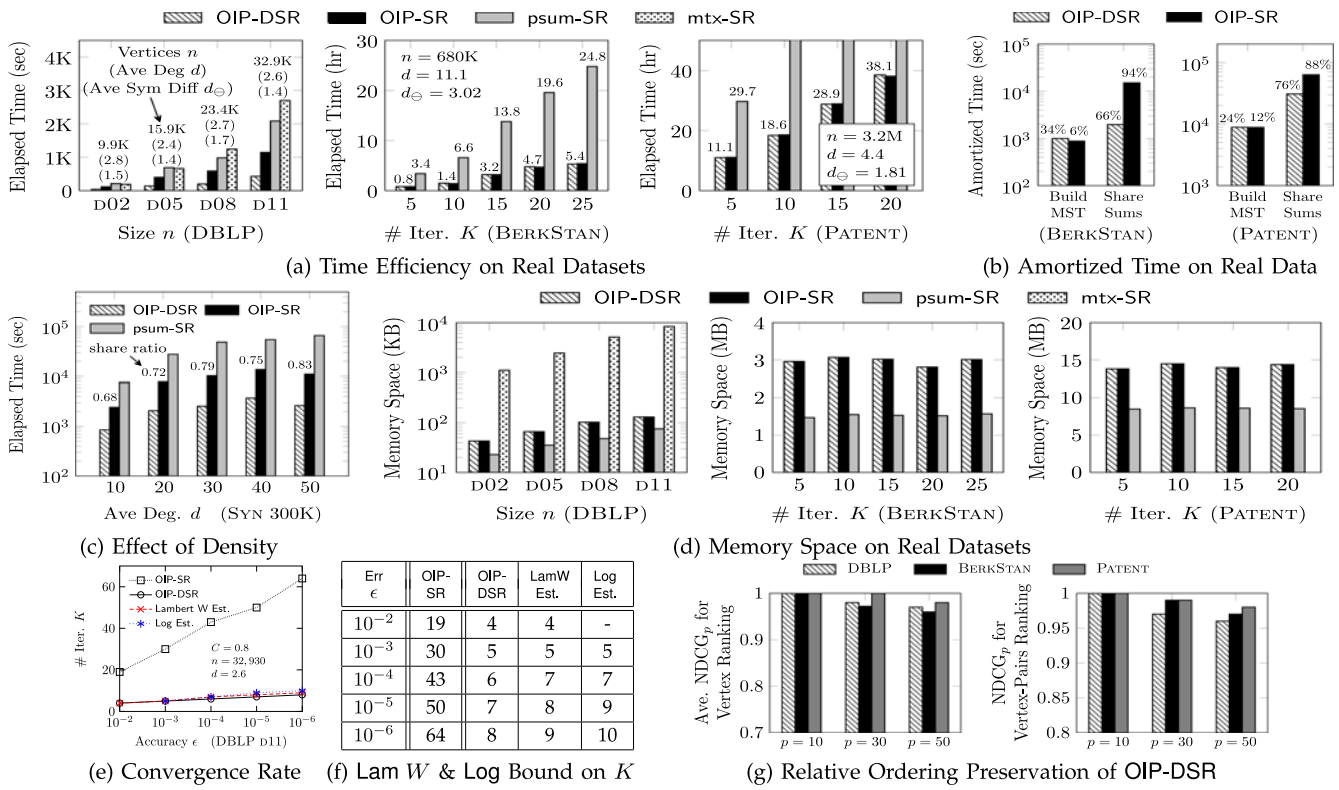
We analyze the time complexity in Appendix E, available in the online supplemental material.

6 Empirical Evaluation

6.1 Experimental Setting

Data sets. For the basic SimRank model, we use three real data sets (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 data sets (Course and IMDB) and one syntectic bipartite data set (SynBI).

The sizes and detailed descriptions of these data sets are depicted in Appendix G, available in the online supplemental material.



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

(f) Lam W & Log Bound on K

Fig. 4. Performance evaluation of OIP-SR and OIP-DSR on real and synthetic data sets.

Compared algorithms. We implement seven algorithms via Visual C++ 8.0. (1) OIP-DSR, our differential SimRank of Eq. (16)[6] in conjunction with partial sums sharing. (2) OIP-SR, our basic SimRank using partial sums sharing. (3) psum-SR [18], without partial sums sharing. (4) mtx-SR [15], 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. (21). (7) MSR [12, Section 4.3.1], the original iterative bipartite Minimax SimRank variation.

We set the following default parameters, as used in [18]: $C = 0.6, \epsilon = 0.001$ (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 [18].

Evaluation metrics. To assess ranking results on real data, we used *Normalized Discounted Cumulative Gain* (NDCG) [15]. The NDCG at rank position p is defined to be $\text{NDCG}_p = \frac{1}{\text{IDCG}_p} \sum_{i=1}^{p} (2^{\text{rank}_i} - 1)/\log_2(1+i)$, 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.

To test the relative order preservation of OIP-DSR relative to OIP-SR, we choose the ranking of OIP-SR as the "ideal" relevance (a baseline), and the ranking of OIP-DSR as the graded relevance $rank_i$ for NDCG_p. Thus, the resulting NDCG_p can reflect the difference of the relative order between OIP-DSR and OIP-SR.

We used a machine powered by a Quad-Core Intel i5 CPU (3.10 GHz) with 16 GB RAM, using Windows 7.

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, Fig. 4a compares 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

6. In the previous conference version [19], OIP-DSR is our differential SimRank of Eq. (15), which requires more memory space for storing the intermediate results.

partial sums sharing. It can be seen that after computing the MST, the sizes of the average symmetric difference $d_{\ominus}$ relative to d are reduced more dramatically on BERKSTAN and PATENT than that on DBLP. Thus, the speedups of our methods on BERKSTAN and PATENT is far more obvious.

Fig. 4b further shows the amortized time for each phase of OIP-SR and OIP-DSR on BERKSTAN and PATENT data (given $\epsilon = 0.001$), 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 is consistent with our complexity analysis in Proposition 5. (2) "Building MST" always takes up larger portions (34 percent on BERKSTAN, and 24 percent on PATENT) in the total time of OIP-DSR, than those (6 percent on BERKSTAN, and 12 percent on PATENT) in the total time of OIP-SR. This becomes more evident on various data sets 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 3 to 15 M on the synthetic data, Fig. 4c reports the impact of graph density (ave. in-degree) on CPU time, where y-axis is in the log scale. Here, share ratio is defined as $1 - \frac{d+(n-1)d_{\ominus}}{nd} = \frac{n-1}{n}(1 - \frac{d_{\ominus}}{d})$, where $d_{\ominus}$ is the average size of symmetric differences (ave. transition costs) for all pairs of in-neighbor sets. A larger share ratio means that in-neighbor sets of a graph have many common vertices for sharing (thus with a smaller $d_{\ominus}$). The results show that (1) for $\epsilon = 0.001$, OIP-DSR significantly outperforms psum-SR by at least one order of magnitude as m increases. On average, OIP-SR achieves 5X speedups. (2) 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 two orders of magnitude speedup for $d = 50$. (3) When d increases from 40 to 50, there is a decreasing tendency in the elapsed time for both OIP-DSR and OIP-SR. This is because in our methods, more partial sub-summations can be shared for later reuse even though the graph density d is increased, as opposed to psum-SR whose time complexity is proportional to d and n. Thus, for the fixed number of vertices in a graph, the performance of our methods mainly hinges on the share ratios among common partial sub-summations (which increases from 0.75 $(d = 40)$ to 0.83 $(d = 50)$). The more share ratios, the more time can be reduced.

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.

Fig. 4d 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 is consistent with our complexity analysis in Section 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. (16) does not need to memoize the auxiliary $\mathbf{T}_k$ in Eq. (15). (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.

Exp-3: convergence rate. 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 data sets are similar.

By varying ϵ from 10^{-2} to 10^{-6}, Figs. 4e and 4f 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 eight iterations, whereas the convergence of OIP-SR in this case becomes sluggish, yielding over 60 iterations. This is consistent with our observation in Proposition 8 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 1 and 2, 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. We next analyze the relative order of similarities from OIP-DSR on real data sets (DBLP, BERKSTAN, and PATENT). On every data set, relative order preservations for both node and node-pairs ranking are evaluated, as shown in Fig. 4g. For node ranking, we fix a vertex a as a given query, and compute the average NDCG_p of OIP-DSR relative to OIP-SR via similarities $s(a, \star)$ from the top-p query perspective. For query selection, we sort all the vertices in order of their degree into four 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. The results are shown in left Fig. 4g. For node-pairs ranking, we find the NDCG_p of OIP-DSR relative to OIP-SR from SimRank scores $s(\star, \star)$ of the top-p similar pairs, as illustrated in right Fig. 4g. The results for $p = 10, 30, 50$ show that OIP-DSR can perfectly maintain the relative order of the similarity scores produced by OIP-DSR with only $<0.8\%$ loss in NDCG_{30} and NDCG_{50} on average for all the data sets. For $p = 10$ (i.e., top-10 node and node-pair queries), OIP-DSR produces exactly the same results of OIP-SR on each dataset. Thus, we can gain a lot in speedup from OIP-DSR while suffering little loss in quality.

A case study for qualitative ranking results on real data is also provided in Appendix H, available in the online supplemental material.

Exp-5: minimax SimRank variation. Finally, we evaluate the time and memory of max-MSR against the baseline psum-MSR and MSR on bipartite real COURSE and IMDB, and synthetic 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 100 K vertices, and vary the

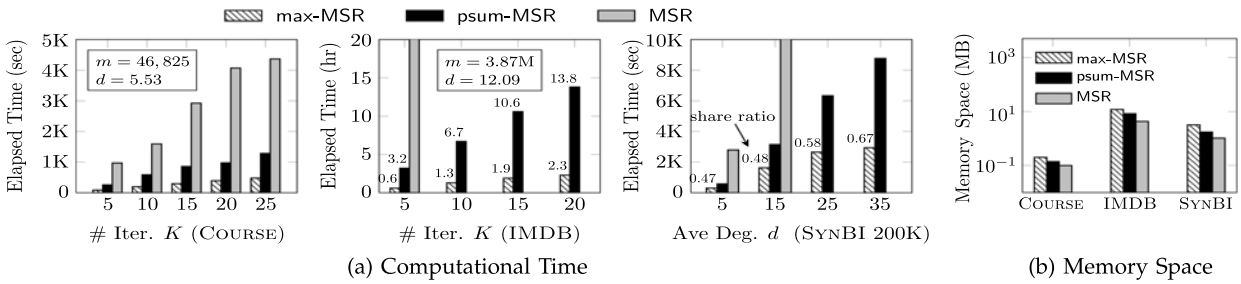


Fig. 5. Performance evaluation of bipartite SimRank variation max-MSR on real and synthetic data sets.

average out-degree from 5 to 35. The results are reported in Fig. 5a. (1) In all the cases, max-MSR is always the fastest, and psum-MSR the second, both of which 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 (0.6 hr) is 5.15X faster than psum-MSR (3.2 hr); however, it takes too long time for MSR to finish the computation within one day. Hence, we stop iterating for MSR after $K \geq 5$ iterations on IMDB and $K \geq 15$ on SYNBI. (2) The graph density has a huge impact on the speedup of max-MSR. The denser a graph, the more likely 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 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 are also consistent with this observation—the share ratio increases w.r.t. the growing average out-degree of the synthetic graph.

The memory space of these Minimax SimRank variations on real and synthetic data sets are evaluated in Fig. 5b. 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 (0.2 M) is slightly higher than psum-MSR (0.14 M) and MSR (0.10 M). 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.

7 CONCLUSIONS

We proposed efficient methods to speed up the computation of SimRank on large networks and bipartite domains. First, we leveraged a novel clustering approach to optimize partial sums sharing. By eliminating duplicate computational efforts among partial summations, an efficient algorithm was devised to greatly reduce the time complexity of SimRank. Second, 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. Third, 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 experiments on both real and synthetic data sets 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 is 5X-12X faster than that of the baselines.

ACKNOWLEDGMENTS

Xuemin Lin is currently supported by NSFC61232006, NSFC61021004, ARC DP140103578 and DP120104168. Wenjie Zhang is supported by ARC DE120102144 and DP120104168.

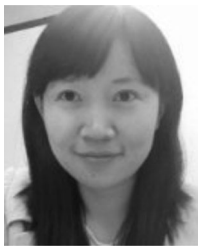
REFERENCES

[1] I. Antonellis, H. G. Molina, and C. Chang, "SimRank++: Query rewriting through link analysis of the click graph," *Proc. VLDB Endowment*, vol. 1, pp. 408-421, 2008.
[2] U. M. Ascher and L. R. Petzold, *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. Philadelphia, PA, USA: SIAM, 1998.
[3] P. Berkhin, "Survey: A survey on PageRank computing," *Internet Math.*, vol. 2, pp. 73–120, 2005.
[4] G. Buehrer and K. Chellapilla, "A scalable pattern mining approach to web graph compression with communities," in *Proc. Int. Conf. Web Search Data Mining*, 2008, pp. 95–106.
[5] J. Cho and S. Roy, "Impact of search engines on page popularity," in *Proc. 13th Int. Conf. World Wide Web*, 2004, pp. 20–29.
[6] D. Fogaras and B. Rácz, "Scaling link-based similarity search," in *Proc. 14th Int. Conf. World Wide Web*, 2005, pp. 641–650.
[7] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka, "Efficient search algorithm for SimRank," in *Proc. IEEE Int. Conf. Data Eng.*, 2013, pp. 589–600.
[8] H. N. Gabow, Z. Galil, T. H. Spencer, and R. E. Tarjan, "Efficient algorithms for finding minimum spanning trees in undirected and directed graphs," *Combinatorica*, vol. 6, pp. 109–122, 1986.
[9] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA, USA: Freeman, 1979.
[10] M. Hassani, "Approximation of the Lambert W function," *RGMIA Res. Rep. Collection*, vol. 8, pp. 1–2, 2005.
[11] G. He, H. Feng, C. Li, and H. Chen, "Parallel SimRank computation on large graphs with iterative aggregation," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2010, pp. 543–552.

[12] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *Proc. 8th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2002, pp. 538–543.
[13] R. Kumar, J. Novak, and A. Tomkins, "Structure and evolution of online social networks," in *Proc. 12th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2006, pp. 611–617.
[14] P. Lee, L. V. Lakshmanan, and J. X. Yu, "On Top-*k* structural similarity search," in *Proc. IEEE Int. Conf. Data Eng.*, 2012, pp. 774–785.
[15] 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 *Proc. 13th Int. Conf. Extending Database Technol.*, 2010, pp. 465–476.
[16] P. Li, H. Liu, J. X. Yu, J. He, and X. Du, "Fast single-pair SimRank computation," in *Proc. SIAM Int. Conf. Data Mining*, 2010, pp. 571–582.
[17] X. Lin, "On the computational complexity of edge concentration," *Discr. Appl. Math.*, vol. 101, no. 1–3, pp. 197–205, 2000.
[18] D. Lizorkin, P. Velikhov, M. N. Grinev, and D. Turdakov, "Accuracy estimate and optimization techniques for SimRank computation," *VLDB J.*, vol. 19, no. 1, pp. 45–66, 2010.
[19] W. Yu, X. Lin, and W. Zhang, "Towards efficient SimRank computation on large networks," in *Proc. IEEE 29th Int. Conf. Data Eng.*, 2013, pp. 601–612.

Weiren Yu received the PhD degree from the School of Computer Science and Engineering, University of New South Wales, Australia, in 2013. He is currently a postdoctoral research associate at the Department of Computing, Imperial College London. His current research interests include graph database, data mining, and link analysis. He received two Canon Information Systems Research Australia (CiSRA) Best Research Paper Awards in 2013 and 2014, one "One of the Best Papers of ICDE" in 2013, and three Best (Student) Paper Awards at APWEB 2010, WAIM 2010, and WAIM 2011, respectively. He is a member of the IEEE and the ACM, and was an active reviewer for many CS conferences and journals.

Xuemin Lin received the BSc degree in applied math from Fudan University in 1984, and the PhD degree in computer science from the University of Queensland, in 1992. He is currently a professor in the School of Computer Science and Engineering, University of New South Wales. He has been the head of database research group at UNSW since 2002, and a concurrent professor at East Normal University since 2009. Before joining UNSW, he held various academic positions at the University of Queensland and the University of Western Australia. During 1984–1988, he studied for a PhD degree in applied math at Fudan University. He is currently an associate editor of the *ACM Transactions on Database Systems*, an associate editor of the *IEEE Transactions on Knowledge and Data Engineering*, and an associate editor of the *World Wide Web Journal*. His current research interests include data mining, data streams, distributed database systems, spatial database systems, web databases, and graph visualization.



Wenjie Zhang received the PhD degree in computer science and engineering in 2010 from the University of New South Wales. She is a currently a lecturer in the School of Computer Science and Engineering, University of New South Wales, Australia. Since 2008, she has published more than 30 papers in SIGMOD, SIGIR, VLDB, ICDE, *TODS*, *TKDE*, and *VLDBJ*. She received the Best (Student) Paper Award of National DataBase Conference of China 2006, APWeb/WAIM 2009, Australasian Database Conference 2010 and DASFAA 2012, and also co-authored one of the best papers in ICDE 2010, ICDE 2012, DASFAA 2012, and ICDE 2013. In 2011, she received the ARC Discovery Early Career Researcher Award. She is currently supported by ARC DE120102144 and DP120104168.

Julie A. McCann is currently a professor of Computer Systems at Imperial College. Her research centers on highly decentralized and self-organizing scalable algorithms for spatial computing systems. She leads both the AESE group and the Intel Research Institute for Sustainable Cities, and is currently working with NEC and others on substantive smart city projects. She has received significant funding through bodies such as the UKs EPSRC, TSB and NERC as well as various international funds, and is an elected peer for the EPSRC. She has actively served on, and chaired, many conference committees and is currently associative editor for the *ACM Transactions on Autonomous and Adaptive Systems*. She is a member of the IEEE and the ACM as well as a chartered engineer, and was elected as a fellow of the BCS in 2013.