Received September 3, 2021, accepted October 21, 2021, date of publication October 26, 2021, date of current version November 2, 2021.

Digital Object Identifier 10.1109/ACCESS.2021.3123114

JacSim*: An Effective and Efficient Solution to the Pairwise Normalization Problem in SimRank

MASOUD REYHANI HAMEDANI[1] AND SANG-WOOK KIM[2], (Member, IEEE)
[1]BK21PLUS Program for Advanced AI Research and Education, Department of Computer Science, Hanyang University, Seoul 04763, South Korea
[2]Department of Computer Science, Hanyang University, Seoul 04763, South Korea

Corresponding author: Sang-Wook Kim (wook@hanyang.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) funded by the Korean Government [Ministry of Science and ICT (MSIT)] under Grant NRF-2020R1A2B5B03001960.

ABSTRACT Despite the fact that SimRank has been successfully applied to various applications as a link-based similarity measure, it suffers from a counter-intuitive property called a *pairwise normalization problem*; JacSim is a powerful variant of SimRank that alleviates this problem. In this paper, we first point out three existing drawbacks of JacSim and then propose *JacSim** to effectively *solve* them; JacSim* *exploits* those paths neglected by JacSim in similarity computation, its matrix form provides the *exact* similarity scores while *not* being sensitive to the number of node-pairs with common neighbors, and it has simpler, easier to understand, and easier to implement formulas in *both* iterative and matrix forms than those of JacSim. We conduct extensive experiments with *eight* real-world datasets to evaluate *both* the accuracy and performance of JacSim* in comparison with those of JacSim. Our experimental results demonstrate that JacSim* shows *better* accuracy than JacSim and the JacSim* matrix form is *dramatically* faster than its own iterative form and also than the two forms of JacSim with *all* datasets.

INDEX TERMS Link-based similarity, pairwise normalization problem, similarity computation, SimRank.

I. INTRODUCTION

In many domains such as social networks, citation networks, bio-medical drug molecules, and the World Wide Web, *graphs* are widely used to encode relational structures where nodes represent objects and links do their relationships in the domain [1]–[3]. In a wide range of applications such as recommender systems, spam detection, web page ranking, and social network analysis, computing accurate similarity among nodes based on the graph structure is a fundamental task [1], [3]. Toward this end, various link-based similarity measures (in short, similarity measures) such as SimRank [4] and its variants [1], [5]–[7] have been proposed in the literature. The philosophy behind SimRank is that two objects are similar if they are related to similar objects and any object is most similar to itself [4]. SimRank *recursively* computes the similarity between two nodes a and b as the *average* of similarity between *all* possible pairs of neighbors pointing to a and b (i.e., in-neighbors) where the similarity between a node and itself is defined as one (i.e., the base case of the recursion); it is called the *pairwise normalization* [1].

The associate editor coordinating the review of this manuscript and approving it for publication was Sathish Kumar.

It is worth to note that to compute the similarity between two nodes, some existing similarity measures such as Struct-Sim [8] exploit the *roles* of nodes in the graph based on the *automorphism equivalent* property; however, SimRank and its variants compute the similarity score of a pair of nodes by exploiting their neighbors (i.e., in-link paths) *regardless* of their roles in the graph. The *graph similarity learning* methods compute the similarity between *two graphs* by applying learning techniques (e.g., graph embedding methods) [9]–[11], while similarity measures compute the similarity between two nodes in a *single* graph. Graph embedding methods exploit the graph structure to represent each node in the graph as a *low-dimensional* vector [12], [13], and then the similarity of two nodes can be computed by applying vector-based measures (e.g., Cosine and Euclidean distance [14]) to their corresponding vectors [8], [15]; on the contrary, similarity measures *directly* exploit the graph structure to compute the similarity of nodes. It has been shown that similarity measures are better than the graph embedding methods to compute nodes similarity [8], [15].

Although SimRank has been successfully applied to many applications such as clustering [16], citation analysis [17], [18], query rewriting [19], k-nearest neighbor search [20]–[22], and link prediction [23], it suffers from a counter-intuitive property raised by the pairwise normalization where a *more number of common in-neighbors may adversely affect the similarity score of a pair of nodes* [1], [5], [6], [24]; it is called the *pairwise normalization problem* [1]. In the literature, different variants of SimRank have been proposed to alleviate this problem. MatchSim [5] exploits only the pairs of similar (i.e., matched) in-neighbors instead of considering all possible pairs of in-neighbors. PSimRank [24], C-Rank [6], and JacSim [1] employ Jaccard coefficient (i.e., Jaccard) [14] along with the pairwise normalization to address the problem; PSimRank and C-Rank behave closely but they apply different normalization techniques. On contrary to PSimRank and C-Rank, JacSim avoids redundancy in computation and assigns an importance factor to the two scores computed based on Jaccard and the pairwise normalization; it has been shown that JacSim significantly outperforms SimRank, PSimRank, C-Rank, and MatchSim in terms of *both* accuracy and performance (i.e., execution time) [1].

As such, JacSim is an excellent variant of SimRank that successfully alleviates the pairwise normalization problem. In this paper, we first point out the following drawbacks of JacSim: 1) JacSim does *not* exploit some paths in the graph, which incurs *limitations* in accurate similarity computation; 2) the JacSim matrix form provides *approximate* similarity scores, thereby providing *lower* accuracy than that of its iterative form; 3) although the JacSim matrix form is significantly faster than its iterative form, it is still *slow* even with small graphs if the graph contains a large number of node-pairs with common in-neighbors. Then, we propose *JacSim** that *not* only effectively solves the above three drawbacks but also preserves the JacSim philosophy in similarity computation to solve the pairwise normalization problem. JacSim* *exploits* those paths neglected by JacSim in similarity computation, its matrix form provides the *exact* similarity scores and *identical* accuracy to that of the iterative form, and the JacSim* matrix form is composed of only matrix-based operations, thereby *not* being sensitive to the number of node-pairs with common in-neighbors. We conduct extensive experiments with *eight* real-world datasets to evaluate *both* the accuracy and performance of our JacSim* in comparison with those of JacSim. Our experimental results with *all* datasets demonstrate that JacSim* *improves* the accuracy of JacSim and the JacSim* matrix form is *dramatically* faster than its own iterative form and also than the both forms of JacSim.

Our contributions in this paper are summarized as follows:

- We propose JacSim* that *exploits* the paths neglected by JacSim in similarity computation while solving the pairwise normalization problem.
- We propose a matrix form for JacSim*, which is *dramatically* faster than its own iterative form and the both forms of JacSim while providing the *exact* similarity scores and *not* being sensitive to the number of node-pairs with common in-neighbors in the graph.

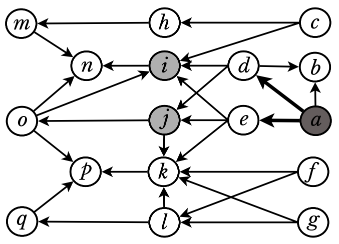


FIGURE 1. A sample graph.

- JacSim* has formulas in *both* iterative and matrix forms simpler, easier to understand, and easier to implement than JacSim.
- We conduct extensive experiments with *eight* real-world datasets to validate the accuracy and performance of our JacSim* in comparison with JacSim.

The remain of this paper is organized as follows. In Section II, we provide some preliminaries about the pairwise normalization problem, JacSim, and its drawbacks. In Section III, we present our proposed similarity measure, JacSim*, in details. Section IV explains the experimental settings and analyzes the results of our experiments. In Section V, we conclude and summarize the paper.

II. PRELIMINARIES

In this section, we provide brief explanations of the pairwise normalization problem, JacSim, and its drawbacks.

A. PAIRWISE NORMALIZATION PROBLEM

In spite of the current success of SimRank, it suffers from the pairwise normalization problem, which is a counter-intuitive property of the pairwise normalization where *more* number of common in-neighbors *may adversely* affect the similarity score of a pair of nodes [1], [5], [6], [24]. Consider the sample graph in Fig. 1; nodes h and i have *one* common direct in-neighbor (i.e., c), while nodes k and l have *two* of them (i.e., f and g). Therefore, the similarity score of node-pair (k, l) should be intuitively higher than that of node-pair (h, i); however, SimRank assigns a lower similarity score to (k, l) (i.e., 0.16) than that of (h, i) (i.e., 0.20). The same circumstance is observed for node-pairs (m, n) and (p, q) each of which do *not* have any common direct in-neighbors; m and n have one common *indirect* in-neighbor (i.e., c), while p and q have two of them (i.e., f and g). It means the similarity score of (p, q) should be higher than that of (m, n); however, the SimRank score of the former node-pair (i.e., 0.042) is lower than that of the latter one (i.e., 0.053).[1]

B. JACSIM

JacSim [1] is a powerful variant of SimRank that alleviates the pairwise normalization problem by employing both Jaccard and the pairwise normalization. Suppose that $G = (V, E)$ is

[1]The SimRank scores are computed by (4) in [4] where C is set as 0.8.

an unweighted and directed[2] graph where V is a set of nodes, $E \in V \times V$ is a set of links among nodes, and I_a denotes a set of nodes *directly* pointing to node a (i.e., direct in-neighbors); JacSim computes the similarity score of a node-pair (a, b), $S(a, b)$, as follows. If $a = b$, then $S(a, b) = 1$; if $a \neq b$ and $I_a = \emptyset$ or $I_b = \emptyset$, then $S(a, b) = 0$; otherwise $S(a, b)$ is calculated by the following recursive formula:

$$S(a,b) = C \cdot \Bigg(\alpha \cdot \frac{|I_a \cap I_b|}{|I_a \cup I_b|} + \frac{(1-\alpha)}{|I_a||I_b| - |I_a \cap I_b|^2} \cdot \Big(\sum_{i \in I_a - I_b} \sum_{j \in I_b} S(i,j) + \sum_{i \in I_b - I_a} \sum_{j \in I_a \cap I_b} S(i,j) \Big)\Bigg) \quad (1)$$

where the left and right sides of + operator in the main parenthesis computed by Jaccard and the pairwise normalization are referred to as the *J-score* and *P-score*, respectively. $\alpha \in (0, 1]$ is an important factor to control the degree of importance of these two scores in similarity computation and $C \in (0, 1)$ is a damping factor.

Consider our sample graph in Fig. 1 again. As explained before, the SimRank score of node-pair (h, i) with one common direct in-neighbor is higher than that of node-pair (k, l) with two common direct in-neighbors due to the pairwise normalization problem; however, JacSim assigns *higher* similarity score to (k, l) (i.e., 0.128) than that of (h, i) (i.e., 0.080). On the contrary to SimRank, JacSim also assigns higher similarity score to (p, q) (i.e., 0.0204) than that of (m, n) (i.e., 0.0128).[3]

C. JACSIM DRAWBACKS

Now, let us point out the existing drawbacks of JacSim as follows.

D1: As observed in (1), to compute the similarity scores for any node-pairs (a, b), JacSim neglects *all* in-neighbor pairs (i, j) where i and j pointing to both a and b (i.e., $i, j \in I_a \cap I_b$) in calculating the P-score since $(I_a - I_b) \cap I_b = \emptyset$ in $\sum_{i \in I_a - I_b} \sum_{j \in I_b} S(i, j)$ part, and $(I_b - I_a) \cap (I_a \cap I_b) = \emptyset$ in $\sum_{i \in I_b - I_a} \sum_{j \in I_a \cap I_b} S(i, j)$ part. As an example, consider nodes i and j in the sample graph in Fig. 1. To compute $S(i, j)$, all node-pairs (d, d), (d, e), (e, d), and (e, e) are neglected in calculating the P-score (i.e., $I_i \cap I_j = \{d, e\}$) where node-pairs (d, d) and (e, e) are ignored to alleviate the pairwise normalization problem (i.e., the similarity score based on the common direct in-neighbors are computed by Jaccard); however, by ignoring node-pairs (d, e) and (e, d), the participation of node a, the common *indirect* in-neighbor pointing to both i and j via nodes d and e, is *not* regarded in computing similarity between i and j. More specifically, JacSim does *not* exploit part of paths in the graph in similarity computation.

D2: The JacSim iterative form represented in (1) *cannot* be directly transformed to a matrix form for the following reason.[4] To calculate the P-score, JacSim partially employs the pairwise normalization on I_a and I_b (i.e., as explained in D1, all in-neighbor pairs (i, j) where $i, j \in I_a \cap I_b$ are ignored and normalization is performed by using the value of $|I_a||I_b| - |I_a \cap I_b|^2$). Therefore, in order to transform the JacSim iterative form to a matrix form, (1) is slightly modified such that the P-score is normalized by the value of $|I_a||I_b|$ instead of $|I_a||I_b| - |I_a \cap I_b|^2$. As a result, the JacSim matrix form does *not* provide the exact JacSim scores (i.e., the approximate scores are computed) and its accuracy is *lower* than that of the iterative form as shown in [1].

D3: The similarity score of a node-pair (a, b) is computed by the JacSim matrix form as follows:

$$S = C \cdot \big(\alpha \cdot J' + (1-\alpha) \cdot (Q^T \cdot S \cdot Q - E)\big) + (1 - C \cdot \alpha) \cdot I$$
$$[\mathrm{E}]_{a,b} = \frac{\sum_{i \in I_a \cap I_b} \sum_{j \in I_a \cap I_b} S(a, b)}{|I_a||I_b|} \quad (2)$$

where $S, J', Q, E, I \in \mathbb{R}^{|V| \times |V|}$, S is a similarity matrix whose entry $[S]_{a,b}$ denotes the similarity score of node-pair (a, b), entry $[J']_{a,b}$ of matrix J' denotes the J-score of (a, b), Q is a *column normalized* adjacency matrix whose entry $[Q]_{a,b} = 1/|I_b|$ if a points to b (i.e., $a \in I_b$); $[Q]_{a,b} = 0$ otherwise. Q^T is a transpose matrix of Q, I is an identity matrix, and E is a matrix whose entry $[E]_{a,b}$ denotes the summation of JacSim scores of all in-neighbor pairs (i, j) where $i, j \in I_a \cap I_b$ normalized by the value of $|I_a||I_b|$. It has been shown that the JacSim matrix form is significantly faster than its iterative form [1].

To accelerate the matrix multiplications in (2), matrices Q and S are represented by the *compressed sparse column* (CSC) storage schema [25]; however, the time complexity of the JacSim matrix form is dominated by computing matrix E. Let $\hbar$ denotes the number of node-pairs (a, b) with common in-neighbors (i.e., $\hbar = |\{(a, b)|I_a \cap I_b \neq \emptyset\}|$), d does the average number of nodes in $I_a \cap I_b$ for all node-pairs (a, b), and k be the number of predefined iterations to compute the similarity scores. As observed in (2), matrix E is computed on each iteration; the time complexity for calculating the entries in E is $O(k\hbar d^2)$, which could be $O(k|V|^4)$ in the worst case; this computation is *slow* when the graph contains a large number of node-pairs with common in-neighbors. More specifically, the execution time of the JacSim matrix form with a *small* graph having a large number of node-pairs with common in-neighbors is *longer* than that with a *large* graph having a small number of such node-pairs. Furthermore, matrix J'(i.e., the J-scores) is computed by the conventional approach (i.e., "*for*" loop), which is expensive with large graphs; the JacSim matrix form is *not* computed by only matrix-based operations since both matrices E and J' are computed by employing "*for*" loops.

[2]For the sake of generality, we regard G as a directed graph since an undirected graph G' can be considered as a directed one where each single link in G' is represented by two links each of which in a different direction.

[3]In (1), C and α are set as 0.8 and 0.4 by following [1].

[4]The complete process of transforming the JacSim iterative form to the matrix form can be found in [1, Section 4].

III. PROPOSED MEASURE: JACSIM*

In this section, we present JacSim* that effectively solves the existing drawbacks of the original JacSim.

A. ITERATIVE FORM

As explained in Section II, JacSim does not exploit some paths in the graph to compute the similarity score of any node-pair (a, b). In order to solve this issue, we propose a novel random walk model as follows: two random walkers r_a and r_b traverse the graph *backward* via in-links (i.e., incoming links to nodes) by starting at a and b (i.e., $a \neq b$), and the two nodes are regarded similar if r_a and r_b meet up at a *common* direct or indirect in-neighbors of a and b; however, the random walkers are supposed to meet up at common *direct* in-neighbors (i.e., r_a visits i and r_b visits j, $i = j$) with the *highest* probability (i.e., 1) where the similarity is computed by Jaccard (i.e., the J-score) or they traverse the graph to meet up at common *indirect* in-neighbors of a and b where the similarity is computed by the pairwise normalization (i.e., the P-score). More specifically, in computing the P-score, we exploit *all* in-neighbor pairs (i, j) where $i \neq j$ (i.e., instead of neglecting $\{(i,j)|i,j \in I_a \cap I_b\}$, only $\{(i,j)|i=j,\, i \in I_a, j \in I_b\}$ are ignored).

JacSim* computes the similarity score of a node-pair (a, b) as follows. If $a = b$, then $S(a, b) = 1$; if $a \neq b$ and $I_a = \emptyset$ or $I_b = \emptyset$, then $S(a, b) = 0$; otherwise $S(a, b) = S'(a, b)$ that is obtained by the following recursive formula:

$$S'(a,b) = C \cdot \left(\alpha \cdot \frac{|I_a \cap I_b|}{|I_a \cup I_b|} + \frac{(1-\alpha)}{|I_a||I_b|} \cdot \sum_{i \in I_a} \sum_{j \in I_b} S'(i,j)\right) \quad (3)$$

where the *base case* of the recursion is $S'(a, b) = 0$ if $a = b$; this base case *guarantees* that *only* those in-neighbor pairs (i, j) where $i \neq j$ are considered in calculating the P-score.

Fig. 2 illustrates *simplified* versions of both JacSim* and JacSim recursive computations to calculate the similarity score of node-pair (i, j) in our sample graph from Fig. 1; in both cases, the circled numbers denote the required recursive calls (i.e., in order of their executions) to compute the P-score of (i, j). JacSim* and JacSim employ seven and five recursive calls to calculate the P-score, respectively; JacSim* exploits two node-pairs (d, e) and (e, d) neglected by JacSim and considers the contribution of node a (i.e., the common indirect in-neighbor of i and j) in similarity computation, thereby assigning the *higher* similarity score (i.e., 0.1984) to (i, j) than the one JacSim does (i.e., 0.1600). In the case of (h, i), (k, l), (m, n), and (p, q), the similarity scores computed by JacSim* are identical to the ones computed by JacSim in our sample graph[5]; JacSim* effectively solves the pairwise normalization problem by preserving the JacSim philosophy in similarity computation.

$I_i = \{c, d, e, o\}, I_j = \{d, e\} \Rightarrow$

JacSim

$$S(i,j) = C \cdot \left(\alpha \cdot \frac{|I_i \cap I_j|}{|I_i \cup I_j|} + (1-\alpha) \cdot \left\{ \overbrace{\underbrace{S(c,d)}_{\textcircled{1}} + \underbrace{S(c,e)}_{\textcircled{2}} + \underbrace{S(o,d)}_{\textcircled{3}} + \underbrace{S(o,e)}_{\textcircled{5}}}^{\text{Pairwise Normalization}} \right\}\right)$$

$$= C \cdot \alpha \cdot \frac{|I_i \cap I_j|}{|I_i \cup I_j|}$$

$\textcircled{1}\,\textcircled{2}\; I_c = \emptyset \Rightarrow S(c,d) = S(c,e) = C \cdot (\alpha \cdot 0 + (1-\alpha) \cdot \{\}) \mapsto$ ***return* (0)**

$\textcircled{3}\; I_o = \{j\}, I_d = \{a\} \Rightarrow S(o,d) = C \cdot \left(\alpha \cdot 0 + (1-\alpha) \cdot \{\overbrace{S(a,j)}^{\textcircled{4}}\}\right) \mapsto$ ***return* (0)**

$\textcircled{4}\; I_a = \emptyset \Rightarrow S(a,j) = C \cdot (\alpha \cdot 0 + (1-\alpha) \cdot \{\}) = 0 \mapsto$ ***return* (0)**

$\textcircled{5}\; S(o,e) = 0 \mapsto$ ***return* (0)** .... similar to $\textcircled{3}$

JacSim*

$$S'(i,j) = C \cdot \left(\alpha \cdot \frac{|I_i \cap I_j|}{|I_i \cup I_j|} + (1-\alpha) \cdot \left\{ \overbrace{\underbrace{S'(c,d)}_{\textcircled{1}} + \underbrace{S'(c,e)}_{\textcircled{2}} + \underbrace{S'(o,d)}_{\textcircled{3}\textcircled{4}} + \underbrace{S'(o,e)}_{\textcircled{5}} + \underbrace{S'(d,e)}_{\textcircled{6}} + \underbrace{S'(e,d)}_{\textcircled{7}}}^{\text{Pairwise Normalization}} \right\}\right)$$

$$= C \cdot \left(\alpha \cdot \frac{|I_i \cap I_j|}{|I_i \cup I_j|} + (1-\alpha) \cdot \left\{2 \cdot C \cdot \alpha \cdot \frac{|I_d \cap I_e|}{|I_d \cup I_e|}\right\}\right)$$

$S'(c,d) = S'(c,e) = S'(o,d) = S'(o,e) = 0$ .... similar to JacSim

$I_d = \{a\}, I_e = \{a\} \Rightarrow$

$\textcircled{6}\; S'(d,e) = C \cdot \left(\alpha \cdot \frac{|I_d \cap I_e|}{|I_d \cup I_e|} + (1-\alpha) \cdot \left\{\underbrace{S'(a,a)}_{0}\right\}\right) \mapsto \boldsymbol{return}\left(C \cdot \alpha \cdot \frac{|I_d \cap I_e|}{|I_d \cup I_e|}\right)$

$\textcircled{7}\; S'(e,d) = C \cdot \left(\alpha \cdot \frac{|I_e \cap I_d|}{|I_e \cup I_d|} + (1-\alpha) \cdot \left\{\underbrace{S'(a,a)}_{0}\right\}\right) \mapsto \boldsymbol{return}\left(C \cdot \alpha \cdot \frac{|I_e \cap I_d|}{|I_e \cup I_d|}\right)$

FIGURE 2. Simplified versions of JacSim* and JacSim computations.

The recursive computation in (3) can be solved by an iteration to a fixed-point for $k = 1, 2, \ldots$ over $S'(a, b)$ as follows. If $a = b$, then $S_k(a, b) = 1$ for any k; if $a \neq b$ and $I_a = \emptyset$ or $I_b = \emptyset$, then $S_k(a, b) = 0$ for any k; otherwise $S_k(a, b)$ is computed by $S'_k(a, b)$:

$$S'_k(a,b) = C \cdot \left(\alpha \cdot \frac{|I_a \cap I_b|}{|I_a \cup I_b|} + \frac{(1-\alpha)}{|I_a||I_b|} \cdot \sum_{i \in I_a} \sum_{j \in I_b} S'_{k-1}(i,j)\right) \quad (4)$$

where $S'_k(a, b) = 0$ if $a = b$; the iterative computation starts with $S'_0(a, b) = 0$ for all node-pairs (a, b).

The JacSim* scores *are* symmetric, bounded, monotonic, unique, and always existent as shown in Appendix A. In Section IV-B2, we show that JacSim* outperforms JacSim in terms of accuracy in similarity computation with all datasets.

B. MATRIX FORM

In this section, we provide a matrix form for our JacSim*. We start by proposing a matrix-based formula for Jaccard, which is employed to compute J-scores; the J-score of a node-pair (a, b) is calculated as follows [1]:

$$J\text{-}score(a,b) = \frac{|I_a \cap I_b|}{|I_a \cup I_b|} \quad (5)$$

We can rewrite (5) as follows:

$$J\text{-}score(a,b) = \frac{|I_a \cap I_b|}{|I_a| + |I_b| - |I_a \cap I_b|} \quad (6)$$

In order to calculate the numerator (i.e., the size of the *intersection* of I_a and I_b) in (6), we provide the following formula:

$$N = A^T \cdot A \quad (7)$$

[5]In both similarity measures, the values of C and α are set as 0.8 and 0.4, respectively, by following [1].

where $A \in \mathbb{R}^{|V|\times|V|}$ is the adjacency matrix of the graph and $N \in \mathbb{R}^{|V|\times|V|}$ is a matrix whose entry $[N]_{a,b}$ indicates the size of the intersection of I_a and I_b.

Lemma 1: In (7), $\forall a, b \in V$, $[N]_{a,b} = |I_a \cap I_b|$.

Proof: Based on the definition of the matrix multiplication, $[N]_{a,b} = \sum_{i=0}^{|v|}[A^T]_{a,i}\cdot[A]_{i,b}$. We know that in the adjacency matrix A, $[A]_{i,j} = 1$ if there is a direct link from i to j (i.e., $i \in I_j$) and $[A^T]_{i,j} = 1$ if there is a direct link from j to i in the graph (i.e., $j \in I_i$). $[A^T]_{a,i}\cdot[A]_{i,b} \neq 0$ if $[A^T]_{a,i} = [A]_{i,b} = 1$, which means $i \in I_a \cap I_b$; therefore, $\sum_{i=0}^{|v|}[A^T]_{a,i}\cdot[A]_{i,b} = |I_a \cap I_b|$.

We obtain the $|I_a|+|I_b|$ part in the denominator of (6) as follows:

$$M = A^T\cdot J + (A^T\cdot J)^T \tag{8}$$

where $J \in \mathbb{R}^{|V|\times|V|}$ is an all-ones matrix (i.e., *all* elements are set as one) and entry $[M]_{a,b}$ in matrix $M \in \mathbb{R}^{|V|\times|V|}$ is identical to the value of $|I_a|+|I_b|$.

Lemma 2: In (8), $\forall a, b \in V$, $[M]_{a,b} = |I_a|+|I_b|$.

Proof: Based on the definition of the matrix multiplication, $[A^T\cdot J]_{a,*} = \sum_{i=0}^{|V|}[A^T]_{a,i}\cdot[J]_{i,*}$. As we know, $[A^T]_{a,i} = 1$ if there is a direct link from i to a in the graph (i.e., $i \in I_a$) and $[J]_{*,*} = 1$; therefore, $[A^T\cdot J]_{a,*} = \sum_{i=0}^{|V|}[A^T]_{a,i}\cdot 1 = |I_a|$. Also, $[A^T\cdot J]_{*,a} = \sum_{i=0}^{|V|}[A^T]_{*,i}\cdot[J]_{i,a}$ and we know that $[A^T]_{*,i} = 1$ if there is a direct link from i to $*$ in the graph (i.e., $i \in |I_*|$); therefore, $[A^T J]_{*,a} = \sum_{i=0}^{|V|}[A^T]_{*,i}\cdot 1 = |I_*|$. More specifically, *all* entries in any row i of matrix $A^T\cdot J$ are *identical* to the value of $|I_i|$ (i.e., $\forall i$, $[A^T\cdot J]_{i,*} = |I_i|$) and each entry $[A^T\cdot J]_{*,j}$ in a column j is identical to $|I_*|$ (i.e., *corresponding* entries in *all* columns are identical where $\forall j$, $[A^T\cdot J]_{*,j} = |I_*|$). As a result, in (8), entry $[M]_{a,b} = [A^T\cdot J]_{a,b} + [(A^T\cdot J)^T]_{a,b} = [A^T\cdot J]_{a,b} + [A^T\cdot J]_{b,a} = |I_a|+|I_b|$.

Now, we can calculate the denominator of (6) as follows:

$$U = M - N \tag{9}$$

where $U \in \mathbb{R}^{|V|\times|V|}$ is a matrix whose entry $[U]_{a,b}$ indicates the size of the *union* of I_a and I_b (i.e., $[U]_{a,b} = |I_a \cup I_b|$).

Finally, we can calculate the J-scores by the following matrix formula:

$$J' = N \oslash U \tag{10}$$

where $\oslash$ denotes the *Hadamard division* (i.e., $[X \oslash Y]_{i,j} = \frac{[X]_{i,j}}{[Y]_{i,j}}$) [26] and entry $[J']_{a,b}$ denotes the J-score of (a, b). However, in the directed graphs, when $I_a = I_b = \emptyset$, the Hadamard division for node-pair (a, b) will be defined as *NaN* (i.e., not a number) since $[N]_{a,b} = [U]_{a,b} = 0$. In order to solve this problem, we rewrite (10) as follows:

$$J' = N \odot (J \oslash U) \tag{11}$$

where $\odot$ denotes the *Hadamard product* (i.e., $[X \odot Y]_{i,j} = [X]_{i,j}\cdot[Y]_{i,j}$) [26]. Note that in calculating $J \oslash U$, we face the *division by zero* problem when $I_a = I_b = \emptyset$ (i.e., $[J \oslash U]_{a,b} = \frac{1}{[U]_{a,b}} = \frac{1}{0} = inf$, positive infinity); however, since $[N]_{a,b} = 0$, the result of the Hadamard product as the final J-score of (a, b) is 0 (i.e., $[J']_{a,b} = 0$ if $I_a = I_b = \emptyset$).

Now, we can propose a matrix form to compute the JacSim* scores in a directed graph. In (3), instead of considering only nodes $i \in I_a$ and $j \in I_b$, we consider *all* nodes in the graph to calculate the P-score of (a, b) as follows:

$$S'(a,b) = C\cdot\Big(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|} + \frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in V}\sum_{j\in V}[A]_{i,a}\cdot S'(i,j)\cdot[A]_{j,b}\Big) \tag{12}$$

we note that $[A]_{i,a} = 1$ if i directly pointing to a (i.e., $i \in I_a$); otherwise, $[A]_{i,a} = 0$. Equation (12) can be rewritten as follows:

$$S'(a,b) = C\cdot\Big(\alpha\cdot[J']_{a,b} + (1-\alpha)\cdot\sum_{i\in V}\sum_{j\in V}\frac{[A]_{i,a}}{|I_a|}\cdot S'(i,j)\cdot\frac{[A]_{j,b}}{|I_b|}\Big) \tag{13}$$

where $\frac{[A]_{i,a}}{|I_a|}$ and $\frac{[A]_{j,b}}{|I_b|}$ are *identical* to $[Q]_{i,a}$ and $[Q]_{j,b}$ (i.e., Q is the column normalized adjacency matrix), respectively; therefore, we provide the following matrix form for JacSim*:

$$\begin{cases} S' = C\cdot\big(\alpha\cdot J' + (1-\alpha)\cdot(Q^T\cdot S'\cdot Q)\big)\wedge I_0 \\ S = S'\vee I \end{cases} \tag{14}$$

where entry $[S]_{a,b}$ in matrix $S \in \mathbb{R}^{|V|\times|V|}$ denotes the JacSim* score of (a, b), $\wedge$ is the *conjunction operator* selecting the minimum operand (i.e., $X \wedge Y = H$, then $[H]_{a,b} = \min\{[X]_{a,b}, [Y]_{a,b}\}$), $\vee$ is the *disjunction operator* selecting the maximum operand (i.e., $X \vee Y = H$, then $[H]_{a,b} = \max\{[X]_{a,b}, [Y]_{a,b}\}$), and in matrix $I_0 \in \mathbb{R}^{|V|\times|V|}$, the diagonal entries are set as 0 and others are set as 1. $\wedge$ and $\vee$ operators set the respective diagonal entries in S' and S as 0 and 1; they guarantee that $S'(a, b) = 0$ if $a = b$ and $S(a, b) = 1$ if $a = b$, respectively.

The recursive formula in (14) can be solved by the following iteration for $k = 1, 2, \ldots$:

$$\begin{cases} S'_k = C\cdot\big(\alpha\cdot J' + (1-\alpha)\cdot(Q^T\cdot S'_{k-1}\cdot Q)\big)\wedge I_0 \\ S_k = S'_k\vee I \\ S_0 = Z \end{cases} \tag{15}$$

the computation is initialized by matrix Z where all entries are set as 0.

Let us clarify the following points about the JacSim* matrix form: 1) as explained step by step, we employed a straightforward mathematical process to transform the JacSim* iterative form to the matrix form *without* applying any changes to the original JacSim* formula in (3); our matrix form provides the *exact* JacSim* scores with no approximation. 2) Contrary to JacSim, our matrix form is represented and calculated *only* by matrix-based operations even in the case of matrix J'. 3) Matrix J' is computed once and reused in all iterations, we have two matrix multiplications in each iteration, and all matrices are represented by the CSC storage

TABLE 1. Some statistics about our datasets.

	$\|V\|$	$\|E\|$	#Labels	Graph Type
Amazon	30,623	97,478	71	undirected
BlogCatalog	10,312	333,982	39	undirected
CoraCitation	23,166	91,500	70	directed
DBLP	21,177	248,131	11	directed
EmailEU	1,005	25,571	42	directed
LiveJournal	11,755	160,046	7,086	undirected
TREC	43,202	347,702	11	directed
Wikipedia	4,777	184,812	40	undirected

schema [25]; let m be the number of non-zero entries in matrix Q, then the time complexity to compute the JacSim* matrix form is $O(km|V|)$.

In Section IV-B3, we show that the JacSim* matrix form is dramatically faster than its iterative form and also than the both forms of JacSim. In Appendix B, we propose JacSim* formulas (i.e., in both forms) that exploit out-neighbors in similarity computation instead of in-neighbors with directed graphs. We note that *both* of the JacSim* formulas proposed in Section III and Appendix B can be equivalently applied to undirected graphs.

IV. EXPERIMENTAL EVALUATION

In this section, we evaluate the accuracy and performance of our JacSim* in comparison with those of JacSim.

A. EXPERIMENTAL SETTINGS

We employ *eight* real-world datasets as follows; Table 1 shows some statistics of our datasets:

Amazon [27] is a products co-purchasing graph collected by crawling Amazon website (i.e., if products a and b are frequently co-purchased, there is a link between them in the graph). The node labels denote the product category. To perform reasonable evaluation, we neglected labels with less than ten corresponding nodes; this graph is partially tagged by 71 labels.

BlogCatalog [12] is a graph where nodes represent bloggers and links do their social relationships. The node labels denote blogger interests inferred through the metadata provided by the bloggers; this graph is fully tagged by 39 labels.

CoraCitation [28] is a citation graph where nodes represent academic papers in the area of computer science and links do citation relationships among papers. The node labels denote the paper's topic (e.g., reinforcement learning); this graph is fully tagged by 70 labels.

DBLP [1] is a citation graph of papers in the areas of data mining and databases published in 2006 and earlier. The node labels denote the papers research topics created based on a famous data mining book [29] where the papers relevant to a chapter's research topic have been grouped together in the bibliographic section of the chapter; the graph is partially tagged by 11 labels corresponding to 11 chapters of the book.

EmailEU [30] is a graph constructed based on the email communication data of a European research institution (i.e., if member a sent at least one email to member b, there is a link from a to b in the graph). The node labels denote the working department of the member; this graph is fully tagged by 42 labels.

LiveJournal [27] is a graph representing social relationships among bloggers. The node labels denote bloggers interests, which are explicitly stated by bloggers themselves. To perform reasonable evaluation, we chose labels with more than ten and less than a hundred nodes; this graph is fully tagged by 7,086 labels.

TREC [1] is a hyperlink graph constructed based on TREC 2003[6] where nodes represent webpages and links do the hyperlinks among them. The node labels indicate the relevant query topic for the webpages, which are created based on LETOR 3.0 [31], a benchmark collection for research on learning to rank for information retrieval, released by Microsoft Research Asia; this graph is partially tagged by 11 labels.

Wikipedia [32] is a co-occurrence graph of words appearing in the first million bytes of the English Wikipedia dump. The labels represent the inferred Part-of-Speech (POS) tags of words. This graph is fully tagged by 40 labels.

To evaluate the accuracy, we utilize MAP (mean average precision), precision, recall, F-score [14], and PRES [33] as evaluation metrics. In each dataset $\mathcal{D}$, labels are considered as ground truth sets and *every* single node tagged by a label l is used as a *query node* q for a similarity based searching to find top-t ($t = 5, 10, 20, 30$) nodes similar to q as a result set; if a node in the result set is originally tagged by l, it is labeled as *relevant*, otherwise *irrelevant*. For each value of t, we compute average precision (AP), precision, recall, F-score, and PRES for q; we take their average values over *all* the queries tagged by l to get the metrics values for l. Then, we compute the average values of MAP, precision, recall, F-score, and PRES over *all* labels in the dataset for t. Finally, the average values of the five metrics over *all* values of t are regarded as the *final* accuracy for $\mathcal{D}$.

For JacSim, damping factor C is set as 0.8 and importance factor α is set as 0.4 by following [1]. In our experiments, we do *not* consider SimRank, PSimRank, C-Rank, and MatchSim since it has been shown than JacSim significantly outperforms all of those similarity measures in terms of both accuracy and performance [1]. All the experiments were performed on an Intel machine equipped with sixteen 3.60 GHz i9-9900K CPUs, 128 GB RAM, and a 64-bit Fedora Core 33 operating system. All required codes are implemented with Python 3.8.

B. RESULTS AND ANALYSES

In this section, we first perform a parameter tuning for JacSim* and then analyze our experimental results.

1) PARAMETER TUNING

First of all, we note that for the parameter tuning and then accuracy comparison in Section IV-B2, we employ the

[6]https://trec.nist.gov/data.html

TABLE 2. Results of parameter tuning with DBLP and LiveJournal datasets.

DBLP	0.1 (3)	0.2 (3)	0.3 (3)	0.4 (3)	0.5 (5)	0.6 (3)	0.7 (3)	0.8 (3)	0.9 (3)
MAP	0.07442	**0.07532**	0.07487	0.07501	0.07534	0.07546	0.07512	0.07470	0.07443
precision	0.17396	**0.17485**	0.17409	0.17461	0.17473	0.17441	0.17354	0.17245	0.17173
PRES	0.10497	**0.10516**	0.10474	0.10503	0.10493	0.10481	0.10415	0.10354	0.10303
recall	0.15238	**0.15299**	0.15261	0.15291	0.15279	0.15255	0.15159	0.15034	0.14930
F-score	0.15019	**0.15073**	0.15028	0.15069	0.15056	0.15029	0.14936	0.14830	0.14743
LiveJournal	0.1 (3)	0.2 (3)	0.3 (5)	0.4 (5)	0.5 (5)	0.6 (5)	0.7 (5)	0.8 (4)	0.9 (3)
MAP	0.06338	0.06335	**0.06340**	0.06329	0.06322	0.06312	0.06297	0.06282	0.06271
precision	0.13752	0.13756	**0.13761**	0.13757	0.13754	0.13746	0.13734	0.13712	0.13699
PRES	0.08315	0.08315	**0.08319**	0.08315	0.08309	0.0830	0.08286	0.08273	0.08261
recall	0.11995	0.11997	**0.12001**	0.11994	0.11983	0.11969	0.11952	0.11930	0.11913
F-score	0.11358	0.11360	**0.11364**	0.11359	0.11352	0.11340	0.11327	0.11307	0.11291

TABLE 3. Best values of α for all datasets.

	Amazon	BlogCatalog	CoraCitation	DBLP	EmailEU	LiveJournal	TREC	Wikipedia
α	0.2	0.2	0.3	0.2	0.2	0.3	0.3	0.1

TABLE 4. Accuracy of JacSim* and JacSim with undirected graphs.

		MAP	precision	PRES	recall	F-score
Amazon	JacSim* (7)	0.41302	0.29357	0.39521	0.48922	0.31392
	JacSim (6)	0.40638	0.28812	0.38667	0.48075	0.30860
BlogCatalog	JacSim* (2)	0.00298	0.09676	0.00404	0.00604	0.00920
	JacSim (2)	0.00291	0.09390	0.00393	0.00589	0.00886
LiveJournal	JacSim* (3)	0.06335	0.13756	0.08315	0.11997	0.11360
	JacSim (6)	0.06132	0.13334	0.08017	0.11524	0.10850
Wikipedia	JacSim* (3)	0.01499	0.07672	0.02198	0.03852	0.02460
	JacSim (3)	0.01428	0.07451	0.02111	0.03746	0.02392

JacSim* matrix form represented by (15) instead of its iterative form represented by (4) since the matrix form provides exact JacSim* scores and it is dramatically faster than the iterative form as discussed in Section IV-B3. As explained in Section III, JacSim* has two parameters: C, the damping factor, and α, the importance factor; we aim to figure out how the accuracy of JacSim* changes when it is equipped with different values of C and α. Our experimental results with all datasets show that the accuracies of JacSim* equipped with different values of C (i.e., $C \in \{0.4, 0.6, 0.8\}$) are *not* tangible; thus, we set the value of C as 0.8 in accordance to JacSim. To find the best value of α, we conduct extensive experiments as follows. For each dataset $\mathcal{D}$, we set the value of α in (15) as 0.1 to 0.9 in step of 0.1 and evaluate the accuracy of JacSim* on ten iterations for each case (i.e., we totally consider $720 = (8 \times 9 \times 10)$ different cases); the value of α providing the *highest* accuracy is selected as the best value for $\mathcal{D}$. We do not consider $\alpha = 0$ and $\alpha = 1.0$ since in the former case, the similarity scores of any node-pairs (a, b) would be zero on all iterations and in the latter case, the similarity scores of any node-pairs (a, b) are computed based on only J-scores on all iterations where *only* direct in-neighbors of a and b are exploited. As an example, Table 2 shows the results of parameter tuning with DBLP and LiveJournal datasets; the values in the parentheses show the iterations on which the highest accuracy are observed (e.g., the highest accuracy with the DBLP dataset when $\alpha = 0.5$ is observed on iteration 5) and the values in boldface show the highest accuracy.

As observed in Table 2, JacSim* shows the *highest* accuracy with DBLP and LiveJournal datasets when $\alpha = 0.2$ and $\alpha = 0.3$, respectively. Table 3 summarizes the complete results of our parameter tuning where the highest accuracy of JacSim* is observed when α is set as 0.2 or 0.3 with *all* datasets except one case (i.e., the Wikipedia dataset); it means JacSim* is *not* too sensitive to the value of α. For the sake of brevity, we set the value of α as 0.2 with *all* datasets for our experimental evaluations in Sections IV-B2 and IV-B3.

2) ACCURACY COMPARISON

In this section, we evaluate the accuracy of JacSim* in comparison with that of JacSim with our datasets in terms of MAP, precision, PRES, recall, and F-score; in this comparison, we consider the iterative form of JacSim since it shows higher accuracy that its matrix form [1]. Let us start with undirected graphs in Amazon, BlogCatalog, LiveJournal, and Wikipedia datasets; with each dataset, we consider the *best* accuracy of both JacSim* and JacSim observed in ten iterations. Table 4 represents the results of this comparison where the numbers in the parentheses show the iterations on which the best accuracies are observed (e.g., JacSim* and JacSim show their

TABLE 5. Accuracy of JacSim* and JacSim with directed graphs by exploiting in-neighbors.

		MAP	precision	PRES	recall	F-score
CoraCitation	JacSim* (6)	0.01147	0.20505	0.01336	0.01629	0.02675
	JacSim (5)	0.01113	0.19792	0.0130	0.01562	0.02604
DBLP	JacSim* (3)	0.07532	0.17485	0.10516	0.15299	0.15073
	JacSim (4)	0.07274	0.16792	0.10045	0.14609	0.14346
EmailEU	JacSim* (2)	0.20646	0.37029	0.22916	0.29951	0.26175
	JacSim (2)	0.20384	0.36374	0.22438	0.29575	0.25805
TREC	JacSim* (2)	0.01262	0.01977	0.01982	0.03531	0.02342
	JacSim (2)	0.01197	0.01938	0.01921	0.03455	0.02298

TABLE 6. Accuracy of JacSim* and JacSim with directed graphs by exploiting out-neighbors.

		MAP	precision	PRES	recall	F-score
CoraCitation	JacSim* (6)	0.02151	0.38170	0.02481	0.02948	0.05030
	JacSim (6)	0.02082	0.37058	0.02383	0.02845	0.04851
DBLP	JacSim* (1)	0.02209	0.06622	0.03619	0.05735	0.05647
	JacSim (1)	0.02209	0.06622	0.03619	0.05735	0.05647
EmailEU	JacSim* (3)	0.18546	0.32325	0.20110	0.26109	0.22612
	JacSim (4)	0.18399	0.31743	0.19813	0.25914	0.22379
TREC	JacSim* (2)	0.03061	0.02950	0.03147	0.03997	0.03068
	JacSim (2)	0.02970	0.02881	0.03061	0.03837	0.02974

best accuracies on iterations 7 and 6 with the Amazon dataset, respectively). As observed in the table, JacSim* shows *batter* accuracy than JacSim in terms of MAP, precision, PRES, recall, and F-score with *all* datasets; the reason is that JacSim* exploits those paths in the graph neglected by JacSim in similarity computation, thereby providing higher accuracy; thanks to our random walk model explained in Section III-A.

Now, we investigate the accuracy of JacSim* in comparison with that of JacSim with directed graphs in CoraCitation, DBLP, EmailEU, and TREC datasets; first, we exploit *in-neighbors* in similarity computation. In this comparison, we also consider the best accuracy of both similarity measures observed in ten iterations and the corresponding iterations are represented in parentheses as before. Table 5 demonstrates the results of this experiment where our observations are in accordance with those in Table 4; JacSim* outperforms JacSim in terms of five metrics with *all* datasets.

Table 6 presents the results of experiments when *out-neighbors* are exploited in similarity computation with our directed graphs. As observed in the table, JacSim* shows better accuracy than JacSim in terms of MAP, precision, PRES, recall, and F-score with *all* datasets except with the DBLP dataset where the both similarity measures show the *identical* accuracy in terms of all five metrics. The reason is that JacSim* and JacSim have their best accuracy on the first iteration where the similarity scores for any node-pairs (a, b) (i.e., $a \neq b$ and $O_a, O_b \neq \emptyset$) are computed only based on the J-scores. For simplicity, let us explain this issue based on in-neighbors as follows; it is applicable to out-neighbors as well. In the case of JacSim* (refer to Section III-A), on the first iteration, $S_1(a, b) = S'_1(a, b) = \alpha \cdot [J']_{a,b} + \frac{(1-\alpha)}{|I_a||I_b|} \sum_{i \in I_a} \sum_{j \in I_b} S'_0(i, j)$ where $S'_0(i, j) = 0$ for all node-pairs (i, j); thus, $S_1(a, b) = S'_1(a, b) = \alpha \cdot [J']_{a,b}$. In the case of JacSim (refer to Section II), $S_1(a, b) = \alpha \cdot [J']_{a,b} + \frac{(1-\alpha)}{|I_a||I_b| - |I_a \cap I_b|^2} \sum_{i \in I_a - I_b} \sum_{j \in I_b} S_0(i, j) + \sum_{i \in I_b - I_a} \sum_{j \in I_a \cap I_b} S_0(i, j)$ where $S_0(i, j) = 0$ for all node-pairs (i, j) that $i \neq j$ and node-pairs (i, j) that $i = j$ are not considered in the computation; thus, $S_1(a, b) = \alpha \cdot [J']_{a,b}$. Although the values of α are set as 0.2 and 0.4 in JacSim* and JacSim as respectively mentioned in Sections IV-B1 and IV-A, the accuracy of the both cases are same regardless of the value of α, since multiplying identical value $[J']_{a,b}$ by constant α does *not* change the similarity-based ranking of node-pairs regarding any query node.

3) PERFORMANCE COMPARISON

In this section, we evaluate the performance (i.e., execution time) of JacSim* in comparison with that of JacSim as follows. We apply each of the JacSim* iterative form (`JS*-IF`), JacSim* matrix form (`JS*-MF`), JacSim iterative form (`JS-IF`), and JacSim matrix form (`JS-MF`) respectively represented by (4), (15), (1), and (2) to our datasets in ten iterations; we do *not* consider the required time to store the results of similarity computations in files or database. Fig. 3 illustrates the execution times of the above measures with our four undirected graphs; times are represented in minutes and the execution time of each similarity measure is also written on the top of its corresponding bar.

We have the following observations in Fig. 3. 1) Although `JS*-IF` outperforms `JS-IF` in terms of accuracy (refer to Section IV-B2), it is slower than `JS-IF` with all datasets; the reason is that `JS*-IF` exploits more paths in similarity computation than `JS-IF` does as explained in Section III-A.

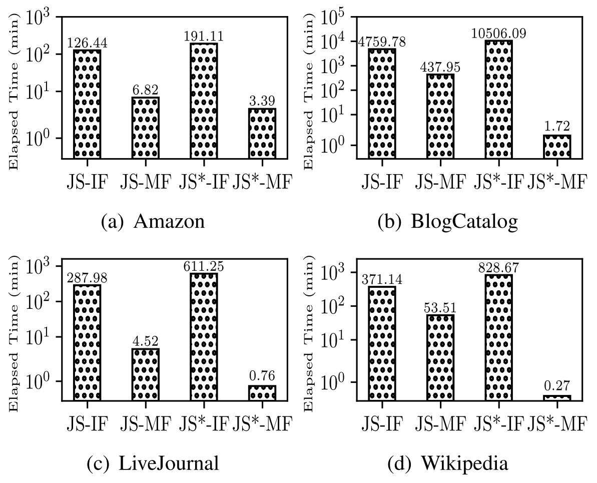


FIGURE 3. **Execution times with undirected graphs.**

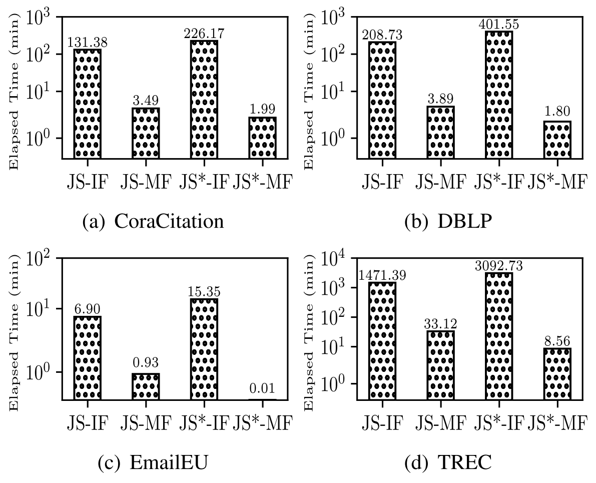


FIGURE 4. **Execution times with directed graphs when exploiting in-neighbors.**

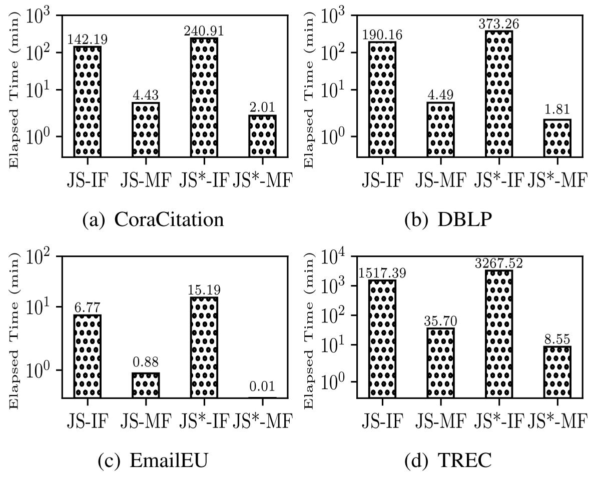


FIGURE 5. **Execution times with directed graphs when exploiting out-neighbors.**

2) JS*-MF is *dramatically* faster than *all* other three similarity measures with *all* datasets since it employs compressed matrices by the CSC storage scheme and *only* matrix-based operations for similarity computations, while both JS*-IF and JS-IF employ the conventional approach (i.e., *"for"* loops) and JS-MF is *not* computed by only matrix-based operations (i.e., matrices E and J' are computed by employing *"for"* loops). 3) JS-MF is *sensitive* to the number of node-pairs with common neighbors in the graph such that it is slower with a small graph having a large number of node-pairs with common in-neighbors than with a large graph having a small number of such node-pairs. As an example, in spite of the fact that BlogCatalog and Wikipedia datasets have *smaller* number of nodes than those of Amazon and LiveJournal (refer to Table 1), the execution times of JS-MF with the two former datasets are *higher* than those with the two latter ones since the number of node-pairs with common neighbors in BlogCatalog (i.e., "32,787,165") and Wikipedia (i.e., "11,015,803") are larger than those in Amazon (i.e.,"780,43") and LiveJournal (i.e., "1,452,366"). On the contrary, the execution time of JS*-MF depends on $|V|$; it shows the highest execution time with the largest undirected dataset (i.e., amazon with "30,000" nodes) and the lowest execution time with the smallest one (i.e., Wikipedia with "4,777" nodes).

Figures 4 and 5 illustrate the results of the performance comparison with the directed graphs where in-neighbors and out-neighbors are exploited, respectively. In these figures, our observations are in accordance with those in Fig. 3. The execution time of JS*-IF is higher than that of JS-IF with all datasets when *any* of in-neighbors and out-neighbors are considered since JS*-IF exploits more paths than JS-IF does in similarity computation. JS*-MF is *dramatically* faster than *all* other similarity measures with all datasets *regardless* of the exploited neighbor type since it employs only compressed matrices along with matrix-based operations for similarity computations. JS-MF is sensitive to the number of node-pairs with common neighbors in the graph *regardless* of the neighbor type. As an example, the DBLP dataset has smaller number of nodes (i.e., "21,177") than that of CoraCitation (i.e., "23,166"); however, the execution times of JS-MF with DBLP based on both in-neighbors and out-neighbors are higher than those with CoraCitaion since the number of node-pairs with common in-neighbors (i.e., "466,990") and out-neighbors (i.e., "2,214,105") in DBLP are larger than those in CoraCitaion (i.e.,"229,306" and "1,100,051", respectively). The execution time of JS*-MF depends on $|V|$; it shows the highest and lowest execution times with TREC as the largest directed dataset (i.e., with "43,202" nodes) and EmailEU as the smallest one (i.e., with "1,005" nodes), respectively, *regardless* of the exploited neighbor type.

In addition, it is worth to note that contrary to JS-MF, JS*-MF shows *same* execution times when exploiting in-neighbors and out-neighbors with each dataset. For example, in the case of the TREC dataset, the execution times of JS-MF with in-neighbors and out-neighbors are 33.12 and 35.70, respectively (i.e., there is 2.58 minutes time difference) and those execution times of JS*-MF are 8.56 and 8.55, respectively (this very small difference is caused by inconsistent system resources such as CPU overload). This again shows that the performance of JS-MF depends on the number of node-pairs with common neighbors in the graph, while that of JS*-MF depends on the actual number of nodes.

V. CONCLUSION

In this paper, we first pointed out the three existing drawbacks of JacSim, a powerful variant of SimRank alleviating the pairwise normalization problem, as follows: 1) JacSim neglects some paths in the graph in similarity computation, which adversely affects its accuracy; 2) the JacSim matrix form provides the approximate similarity scores; thus, it shows lower accuracy than its iterative form, and 3) the JacSim matrix form still suffers from the low performance since it is sensitive to the number of node-pairs with common neighbors in the graph. Then, we proposed JacSim*, which effectively solves the above three issues along with the pairwise normalization problem, it shows higher accuracy than JacSim with eight real-world datasets, its matrix form provides the exact similarity scores and identical accuracy to that of the iterative form while it is dramatically faster than its own iterative form and the both forms of JacSim, its matrix form is not sensitive to the number of node-pairs with common neighbors, and it has more simpler, easier to understand, and easier to implement formulas in both iterative and matrix forms than the original JacSim.

We figured out interesting directions for our future work as follows. We plan to extend JacSim* to compute similarity of nodes in *signed* graphs, where two types of links (i.e., positive and negative) exist. It has been shown that negative links contain additional information, which is beneficial to various tasks such as link sign prediction and node classification in signed graphs [34]. Furthermore, to highly improve the performance of JacSim* with very large graphs (i.e., with millions or billions of nodes), we plan to propose an *acceleration technique* (e.g., partial sums memoization [35]) for our JacSim*; in this case, we need to apply such technique to only matrix multiplications in (15) since matrix J' is computed once and reused in all iterations.

APPENDIX A

In this section, we show that the JacSim* scores are symmetric, bounded, monotonic, unique, and always existent.

(1) **Symmetry**: for any node-pair (a, b), $S(a, b) = S(b, a)$. According to (4), if $a=b$, $S_k(a, b)=S_k(b, a)=1$; if $a\neq b$ and $I_a=\emptyset$ or $I_b=\emptyset$, $S_k(a, b)=S_k(b, a)=0$; otherwise $S_k(a, b)=S'_k(a, b)=S'_k(b, a)=S_k(b, a)$ for all $k\geq 0$.

(2) **Bounding**: for all k, $0\leq S_k(a, b)\leq 1$.

According to (4), if $a=b$, then $S_0(a, b)=S_1(a, b)=\cdots=1$; therefore, $0\leq S_k(a, b)\leq 1$ for all values of k. If $a\neq b$ and $I_a=\emptyset$ or $I_b=\emptyset$, then $S_0(a, b)=S_1(a, b)=\cdots=0$; therefore, $0\leq S_k(a, b)\leq 1$ for all values of k. If $a\neq b$, $I_a\neq\emptyset$, and $I_b\neq\emptyset$, then $S_0(a, b) = S'_0(a, b) = 0$; therefore, $0 \leq S_0(a, b) \leq 1$, which means the property holds for $k = 0$. Now, we assume that the property holds for k, which means $0 \leq S_k(a, b) = S'_k(a, b)\leq 1$; according to this assumption $S'_k(a, b)\geq 0$, thus

$$
\begin{aligned}
S'_{k+1}(a, b) &= C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}+\frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}S'_k(i, j)\right)\\
&\geq C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}+\frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}0\right)\\
&\geq C\cdot\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}
\end{aligned}
$$

where $0\leq\frac{|I_a\cap I_b|}{|I_a\cup I_b|}\leq 1$, $0<C<1$, and $0<\alpha\leq 1$, which means $S'_{k+1}(a, b)=S_{k+1}(a, b)\geq 0$.

According to the assumption $S'_k(a, b)\leq 1$, thus

$$
\begin{aligned}
S'_{k+1}(a, b) &= C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}+\frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}S'_k(i, j)\right)\\
&\leq C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}+\frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}1\right)
\end{aligned}
$$

since $\sum_{i\in I_a}\sum_{j\in I_b}1=|I_a||I_b|$, then $S'_{k+1}(a, b)\leq C\cdot\big(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}+(1-\alpha)\big)$. We know that $\frac{|I_a\cap I_b|}{|I_a\cup I_b|}\leq 1$, which means $S'_{k+1}(a, b)\leq C\cdot\alpha+C\cdot(1-\alpha)=C$; since $0<C<1$, then also $S'_{k+1}(a, b)=S_{k+1}(a, b)\leq 1$.

(3) **Monotonicity**: for every node-pair (a, b), the sequence $\{S_k(a, b)\}$ is non-decreasing as k increases.

If $a = b$, $S_0(a, b) = S_1(a, b) = \cdots = 1$; thus, the property holds. If $a \neq b$ and $I_a = \emptyset$ or $I_b = \emptyset$, $S_0(a, b) = S_1(a, b) = \cdots = 0$; thus, the property holds. If $a \neq b$, $I_a \neq \emptyset$, and $I_b \neq \emptyset$, according to (4), $S_0(a, b) = 0$ and by the bounding property, $0 \leq S_1(a, b) \leq 1$; therefore, $S_0(a, b) \leq S_1(a, b)$, which means for $k = 0$, the property holds. We assume that the property holds for all k where $S_{k-1}(a, b)=S'_{k-1}(a, b)\leq S_k(a, b) = S'_k(a, b)$, which means $S'_k(a, b)-S'_{k-1}(a, b) \geq 0$. Now, we show the property holds for $k+1$ as follows:

$$
\begin{aligned}
&S'_{k+1}(a, b) - S'_k(a, b)\\
&\quad= C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}+\frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}S'_k(i, j)\right)\\
&\qquad -C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}+\frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}S'_{k-1}(i, j)\right)\\
&\quad=\frac{C\cdot(1-\alpha)}{|I_a||I_b|}\left(\sum_{i\in I_a}\sum_{j\in I_b}S'_k(i, j)-\sum_{i\in I_a}\sum_{j\in I_b}S'_{k-1}(i, j)\right)\\
&\quad=\frac{C\cdot(1-\alpha)}{|I_a||I_b|}\left(\sum_{i\in I_a}\sum_{j\in I_b}S'_k(i, j)-S'_{k-1}(i, j)\right)
\end{aligned}
$$

according to the assumptions, $S'_k(a, b)-S'_{k-1}(a, b) \geq 0$ and we already know that $C\cdot(1-\alpha) \geq 0$ and $|I_a||I_b| \geq 0$; therefore, $S'_{k+1}(a, b)–S'_k(a, b)\geq 0$, which means $S_{k+1}(a, b)=S'_{k+1}(a, b)\geq S'_k(a, b)=S_k(a, b)$.

(4) **Existence**: the fixed points $S(*, *)$ of the JacSim* equation always exists.

By the bounding and monotonicity properties, for any node-pairs (a, b), $S'_k(a, b)$ is bounded and non-decreasing as

k increases; a sequence $S'_k(a,b)$ converges to a $\lim S'(a,b) = S(a,b)$ in [0, 1], according to the Completeness Axiom of calculus. $\lim_{k\to\infty} S'_{k+1}(a,b) = \lim_{k\to\infty} S'_k(a,b) = S'(a,b)$ and the limit of a sum is identical to the sum of the limits, therefore

$$\begin{aligned} &S'(a,b) \\ &= \lim_{k\to\infty} S'_{k+1} \\ &= C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|} + \frac{(1-\alpha)}{|I_a||I_b|}\cdot\lim_{k\to\infty}\sum_{i\in I_a}\sum_{j\in I_b} S'_k(i,j)\right) \\ &= C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|} + \frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}\lim_{k\to\infty} S'_k(i,j)\right) \\ &= C\cdot\left(\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|} + \frac{(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b} S'(i,j)\right) = S(a,b) \end{aligned}$$

(5) **Uniqueness**: the solution for the fixed-point $S(*,*)$ is always unique.

Suppose that $S(*,*)$ and $\widehat{S}(*,*)$ are two solutions for the JacSim* equation. Also, for all node-pairs (a,b), let $\delta(a,b) = S(a,b) - \widehat{S}(a,b)$ be the difference between these two solutions. Let $M = \max|\delta(a,b)|$ be the maximum absolute value of all differences observed for node-pairs (a,b) (i.e., $|\delta(a,b)| = M$). We need to prove that $M = 0$. If $a = b$, $M = 0$ since $S(a,b) = \widehat{S}(a,b) = 1$. If $a \neq b$ and $I_a = \emptyset$ or $I_b = \emptyset$, $M = 0$ since $S(a,b) = \widehat{S}(a,b) = 0$. Otherwise, $S(a,b) = S'(a,b)$ and $\widehat{S}(a,b) = \widehat{S}'(a,b)$ are computed by JacSim*. When $\alpha = 1$, $M = 0$ since $S'(a,b) = \widehat{S}'(a,b) = C\cdot\alpha\cdot\frac{|I_a\cap I_b|}{|I_a\cup I_b|}$. When $0 < \alpha < 1$, we have

$$\begin{aligned} \delta(a,b) &= S'(a,b) - \widehat{S}'(a,b) \\ &= \frac{C\cdot(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b} S'(i,j) - \widehat{S}'(i,j) \\ &= \frac{C\cdot(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}\delta(i,j) \end{aligned}$$

thus,

$$\begin{aligned} M = |\delta(a,b)| &= \left|\frac{C\cdot(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}\delta(i,j)\right| \\ &\leq \frac{C\cdot(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b}|\delta(i,j)| \\ &\leq \frac{C\cdot(1-\alpha)}{|I_a||I_b|}\cdot\sum_{i\in I_a}\sum_{j\in I_b} M \\ &= C\cdot(1-\alpha)\cdot M \end{aligned}$$

Since $0 < \alpha < 1$ and $0 < C < 1$, then $0 < C\cdot(1-\alpha) < 1$, which means $M = 0$.

APPENDIX B

In this section, we provide JacSim* formulas that exploit out-neighbors in similarity computation instead of in-neighbors in directed graphs. Let us define O_a as a set of nodes directly pointed to by node a (i.e., direct out-neighbors of a). Then, the similarity score of a node-pair (a,b) by considering out-neighbors is computed as follows by JacSim*. If $a = b$, then $S(a,b) = 1$; if $a \neq b$ and $O_a = \emptyset$ or $O_b = \emptyset$, then $S(a,b) = 0$; otherwise $S(a,b) = S'(a,b)$ that is obtained by the following formula:

$$S'(a,b) = C\cdot\left(\alpha\cdot\frac{|O_a\cap O_b|}{|O_a\cup O_b|} + \frac{(1-\alpha)}{|O_a||O_b|}\cdot\sum_{i\in O_a}\sum_{j\in O_b} S'(i,j)\right) \tag{16}$$

where the base case of the recursive computation is $S'(a,b) = 0$ if $a = b$; this base case guarantees that only those out-neighbor pairs (i,j) where $i \neq j$ are considered in calculating the P-score.

The overall mathematical process to transform the JacSim* iterative form to a matrix form based on out-neighbors is *exactly* similar to the one represented in Section III-B except we exploit out-neighbors instead of in-neighbors. The JacSim* matrix form based on out-neighbors is as follows:

$$\begin{cases} S' = C\cdot\left(\alpha\cdot J' + (1-\alpha)\cdot(Q\cdot S'\cdot Q^T)\right)\wedge I_0 \\ S = S' \vee I \end{cases} \tag{17}$$

where entry $[S]_{a,b}$ in matrix $S \in \mathbb{R}^{|V|\times|V|}$ denotes the similarity score of (a,b), $Q \in \mathbb{R}^{|V|\times|V|}$ is the *row normalized* adjacency matrix whose entry $[Q]_{a,b} = 1/|O_a|$ if b is pointed to by a (i.e., $b \in O_a$); otherwise $[Q]_{a,b} = 0$, and matrix J' is calculated as follows:

$$\begin{aligned} J' &= N \odot (J \oslash (M - N)) \\ N &= A\cdot A^T \\ M &= A\cdot J + (A\cdot J)^T \end{aligned} \tag{18}$$

We note that *both* recursive formulations in (16) and (17) can be solved by the iteration to a fixed-point similar to the ones explained in Section III.

REFERENCES

[1] M. R. Hamedani and S.-W. Kim, "JacSim: An accurate and efficient link-based similarity measure in graphs," *Inf. Sci.*, vol. 414, pp. 203–224, Nov. 2017.

[2] Y. Wang, Z. Wang, Z. Zhao, Z. Li, X. Jian, H. Xin, L. Chen, J. Song, Z. Chen, and M. Zhao, "Effective similarity search on heterogeneous networks: A meta-path free approach," *IEEE Trans. Knowl. Data Eng.*, early access, Aug. 26, 2020, doi: 10.1109/TKDE.2020.3019488.

[3] W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei, "More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks," *Proc. VLDB Endowment*, vol. 7, no. 1, pp. 13–24, Sep. 2013.

[4] G. Jeh and J. Widom, "SimRank: A measure of structural-context similarity," in *Proc. 8th Int. Conf. Knowl. Discovery Data Mining, (SIGKDD)*, Jul. 2002, pp. 538–543.

[5] Z. Lin, M. R. Lyu, and I. King, "MatchSim: A novel similarity measure based on maximum neighborhood matching," *Knowl. Inf. Syst.*, vol. 32, no. 1, pp. 141–166, Jul. 2012.

[6] S.-H. Yoon, S.-W. Kim, and S. Park, "C-Rank: A link-based similarity measure for scientific literature databases," *Inf. Sci.*, vol. 326, pp. 25–40, Jan. 2016.

[7] P. Zhao, J. Han, and S. Yizhou, "P-Rank: A comprehensive structural similarity measure over information networks," in *Proc. 18th ACM Int. Conf. Inf. Knowl. Manage. (CIKM)*, 2009, pp. 553–562.

[8] X. Chen, L. Lai, L. Qin, and X. Lin, "Efficient structural node similarity computation on billion-scale graphs," *VLDB J.*, vol. 30, no. 3, pp. 471–493, Feb. 2021.

[9] G. Ma, N. K. Ahmed, T. L. Willke, and P. S. Yu, "Deep graph similarity learning: A survey," *Data Mining Knowl. Discovery*, vol. 35, no. 3, pp. 688–725, Mar. 2021.

[10] Y. Li, C. Gu, T. Dullien, O. Vinyals, and P. Kohli, "Graph matching networks for learning the similarity of graph structured objects," in *Proc. 36th Int. Conf. Mach. Learn. (ICML)*, 2019, pp. 3835–3845.

[11] Y. Bai, H. Ding, K. Gu, Y. Sun, and W. Wang, "Learning-based efficient graph similarity computation via multi-scale convolutional set matching," in *Proc. 34th AAAI Conf. Artif. Intell.*, 2020, pp. 3219–3226.

[12] J. Qiu, Y. Dong, H. Ma, J. Li, K. Wang, and J. Tang, "Network embedding as matrix factorization: Unifying DeepWalk, LINE, PTE, and node2vec," in *Proc. 11st ACM Int. Conf. Web Search Data Mining (WSDM)*, 2018, pp. 459–467.

[13] Z. Zhao, H. Zhou, C. Li, J. Tang, and Q. Zeng, "DeepEmLAN: Deep embedding learning for attributed networks," *Inf. Sci.*, vol. 543, pp. 382–397, Jan. 2021.

[14] C. Manning, P. Raghavan, and H. Schutze, *Introduction to Information Retrieval*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

[15] M. R. Hamedani and S. W. Kim, "On investigating both effectiveness and efficiency of embedding methods in task of similarity computation of nodes in graphs," *Appl. Sci.*, vol. 11, p. 162, Jan. 2021.

[16] Y. Cai, P. Li, H. Liu, J. He, and X. Du, "S-SimRank: Combining content and link information to cluster papers effectively and efficiently," in *Proc. 4th Int. Conf. Adv. Data Mining Appl. (ADMA)*, 2008, pp. 317–329.

[17] M. R. Hamedani, S.-W. Kim, and D.-J. Kim, "SimCC: A novel method to consider both content and citations for computing similarity of scientific papers," *Inf. Sci.*, vols. 334–335, pp. 273–292, Mar. 2016.

[18] 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 (KDD)*, 2010, pp. 543–552.

[19] I. Antonellis, H. G. Molina, and C. C. Chang, "Simrank++: Query rewriting through link analysis of the clickgraph," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 408–421, Aug. 2008.

[20] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka, "Efficient search algorithm for SimRank," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Apr. 2013, pp. 589–600.

[21] M. Kusumoto, T. Maehara, and K.-I. Kawarabayashi, "Scalable similarity search for SimRank," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2014, pp. 325–336.

[22] Y. Shao, B. Cui, L. Chen, M. Liu, and X. Xie, "An efficient similarity search framework for SimRank over large dynamic graphs," *Proc. VLDB Endowment*, vol. 8, no. 8, pp. 838–849, Apr. 2015.

[23] A. Tsitsulin, D. Mottin, P. Karras, and E. Müller, "VERSE: Versatile graph embeddings from similarity measures," in *Proc. 18th Int. Conf. World Wide Web (WWW)*, 2018, pp. 539–548.

[24] D. Fogaras and B. Rácz, "Scaling link-based similarity search," in *Proc. 14th Int. Conf. World Wide Web (WWW)*, 2005, pp. 641–650.

[25] Y. Saad, *Iterative Methods for Sparse Linear Systems*. Philadelphia, PA, USA: SIAM, 2003.

[26] R. A. H. Han and C. R. Johnson, *Matrix Analysis*, 2nd ed. Cambridge, U.K.: Cambridge Univ. Press, 2013.

[27] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," in *Proc. 12th Int. Conf. Data Mining (ICDM)*, 2012, pp. 745–754.

[28] L. Šubelj and M. Bajec, "Model of complex networks based on citation dynamics," in *Proc. 22nd Int. Conf. World Wide Web (WWW)*, 2013, pp. 527–530.

[29] J. Han, M. Kamber, and J. Pei, *Data Mining: Concepts and Techniques*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2006.

[30] H. Yin, A. R. Benson, J. Leskovec, and D. F. Gleich, "Local higher-order graph clustering," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 555–564.

[31] T. Qin, T.-Y. Liu, J. Xu, and H. Li, "LETOR: A benchmark collection for research on learning to rank for information retrieval," *Inf. Retr.*, vol. 13, no. 4, pp. 346–374, 2010.

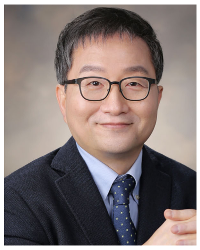
[32] H. Wang, J. Wang, J. Wang, M. Zhao, W. Zhang, F. Zhang, X. Xie, and M. Guo, "GraphGAN: Graph representation learning with generative adversarial nets," in *Proc. 32th AAAI Conf. Artif. Intell.*, 2018, pp. 2508–2515.

[33] W. Magdy and G. J. Jones, "PRES: A score metric for evaluating recall-oriented information retrieval applications," in *Proc. 33rd ACM SIGIR Int. Conf. Res. Develop. Inf. Retr.*, 2010, pp. 611–618.

[34] J. Kunegis, J. Preusse, and F. Schwager, "What is the added value of negative links in online social networks?" in *Proc. 22nd Int. Conf. World Wide Web (WWW)*, 2013, pp. 727–736.

[35] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov, "Accuracy estimate and optimization techniques for simrank computation," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 422–433, 2008.

MASOUD REYHANI HAMEDANI received the B.S. degree in computer science from Shahid Bahonar University, Kerman, Iran, in 2004, the M.S. degree in software engineering from Payame Nour University, Tehran, Iran, in 2009, and the Ph.D. degree in computer and software from Hanyang University, Seoul, South Korea, in 2016. He worked as a Postdoctoral Researcher with Dankook University, Yongin, South Korea, until February 2018. In March 2018, he joined as an Assistant Professor with the Department of Computer and Software, Hanyang University. He is currently working as a Research Assistant Professor. His research interests include data science, similarity computation in social networks, and deep learning.



SANG-WOOK KIM (Member, IEEE) received the B.S. degree in computer engineering from Seoul National University, in 1989, and the M.S. and Ph.D. degrees in computer science from the Korea Advanced Institute of Science and Technology (KAIST), in 1991 and 1994, respectively. From 1995 to 2003, he worked as an Associate Professor with Kangwon National University. In 2003, he joined Hanyang University, Seoul, South Korea, where he is currently a Professor with the Department of Computer Science and Engineering and the Director of the Brain-Korea-21-Plus Research Program. He has been leading a National Research Laboratory (NRL) Project funded by the National Research Foundation, since 2015. From 2009 to 2010, he visited the Computer Science Department, Carnegie Mellon University, as a Visiting Professor. From 1999 to 2000, he worked with the IBM T. J. Watson Research Center, USA, as a Postdoctoral Researcher. He also visited the Computer Science Department, Stanford University, as a Visiting Researcher, in 1991. He is the author of more than 200 papers in refereed international journals and international conference proceedings. His research interests include databases, data mining, multimedia information retrieval, social network analysis, recommendation, and web data analysis. He is a member of the ACM.