AdaSim: A Recursive Similarity Measure in Graphs

Masoud Reyhani Hamedani
masoud@hanyang.ac.kr
Department of Computer Science
BK21PLUS Program for Advanced AI Research and Education, Hanyang University
Seoul, Korea

Sang-Wook Kim*
wook@hanyang.ac.kr
Department of Computer Science
Hanyang University
Seoul, Korea

ABSTRACT

In the literature, various link-based similarity measures such as Adamic/Adar (in short Ada), SimRank, and random walk with restart (RWR) have been proposed. Contrary to SimRank and RWR, Ada is a *non-recursive* measure, which exploits the *local* graph structure in similarity computation. Motivated by Ada's *promising* results in various graph-related tasks, along with the fact that SimRank is a recursive generalization of the co-citation measure, in this paper, we propose *AdaSim*, a recursive similarity measure based on the *Ada philosophy*. Our AdaSim provides *identical* accuracy to that of Ada on the first iteration and it is applicable to *both* directed and undirected graphs. To accelerate our iterative form, we also propose a *matrix form* that is *dramatically faster* while providing the *exact* AdaSim scores. We conduct extensive experiments with *five* real-world datasets to evaluate *both* the effectiveness and efficiency of our AdaSim in comparison with those of existing similarity measures and graph embedding methods in the task of similarity computation of nodes. Our experimental results show that 1) AdaSim significantly *improves* the effectiveness of Ada and *outperforms* other competitors, 2) its efficiency is *comparable* to that of SimRank* while being better than the others, 3) AdaSim is *not* sensitive to the parameter tuning, and 4) similarity measures are *better* than embedding methods to compute similarity of nodes.

CCS CONCEPTS

• **Information systems → Data mining**.

KEYWORDS

link-based similarity, Adamic/Adar, recursive measure, graph structure, pairwise normalization

ACM Reference Format:
Masoud Reyhani Hamedani and Sang-Wook Kim. 2021. AdaSim: A Recursive Similarity Measure in Graphs. In *Proceedings of the 30th ACM International Conference on Information and Knowledge Management (CIKM '21), November 1–5, 2021, Virtual Event, QLD, Australia.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3459637.3482316

*Sang-Wook Kim is the corresponding author.

CIKM '21, November 1–5, 2021, Virtual Event, QLD, Australia

ACM ISBN 978-1-4503-8446-9/21/11…$15.00
https://doi.org/10.1145/3459637.3482316

1 INTRODUCTION

In the literature, various link-based similarity measures (in short, similarity measures) have been proposed such as Adamic/Adar [1], SimRank [13] and its variants (e.g., P-Rank [42], JacSim [9], and SimRank* [40]), and random walk with restart (RWR) [35] and its variants (e.g., BePI [14] and BEAR [15]). Adamic/Adar (in short, Ada) was originally proposed to predict whether a person is associated with another based on their personal homepages by mainly focusing on university homepages; it is based on a philosophy that "*anything from the links and text on a user's homepage to the mailing lists a user subscribes to are reflections of social interactions the user has in the real world*" [1]. To do this, four different items are extracted from each user's homepage: text, in-links (i.e., links from other pages to the page), out-links (i.e., links from the page to other pages), and mailing lists. Then, the similarity between users a and b is computed as follows [1]:

$$S(a,b) = \sum_{shared\ items} \frac{1}{\log(frequency(shared\ item))} \quad (1)$$

More specifically, the *weights* of common items are summed up where the inverse of the log frequency of an item's occurrence is utilized as the item's weight; the items that are unique to a *fewer* number of pages are weighted *higher* than the items commonly occurring in many pages [1]. This strategy is somehow in *accordance* with the philosophy followed by the TF-IFD weighting scheme [23] in the text mining, where two documents are regarded more similar if they commonly contain such terms that are repeated in both documents but rarely appear in other documents.

The interesting idea demonstrated by Equation (1) suggests us a measure to compute the similarity between nodes in graphs where two nodes are regarded *more* similar if they *both* connected to more common nodes, which have *fewer* links in the graph; the *similarity score of a node-pair* (a, b), $S(a, b)$, is then computed as follows [19]:

$$S(a,b) = \sum_{i \in N_a \cap N_b} \frac{1}{\log(|N_i|)} \quad (2)$$

where N_a and N_b denote two sets of nodes *directly* connected to a and b, respectively; Ada exploits *only* the *direct neighbors* of a and b, which means Ada exploits the graph structure *locally* in similarity computation.

SimRank [13] and its variants have attracted significant attention in the literature [9, 18, 40]. The philosophy behind SimRank is that two nodes are similar if they are referenced (i.e., linked) by similar nodes, and a node is most similar to itself [13]. In SimRank, $S(a, b)$ is *recursively* computed as the *average* similarity scores of *all* possible pairs of neighbors pointing to a and b where the base case of this recursive computation is $S(a, a) = 1$ for any nodes a [13]; this is

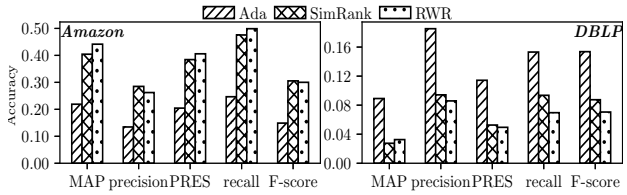


Figure 1: Accuracy comparison of Ada, SimRank, and RWR.

called a *pairwise normalization* approach [9]. SimRank not only considers the direct neighbors of a and b but also considers their indirect neighbors by exploiting the graph structure *globally* in similarity computation. Random walk with restart (RWR) [35] and its variants are another well known family of similarity measures in the literature. In RWR, a random walker starting from a node a visits nodes in the graph *iteratively* via links with probabilities proportional to the link weights (i.e., random walk) or it returns to node a with a predefined probability (i.e., restart); the steady-state probability that the random walker stays at a node b is regarded as $S(a, b)$. RWR exploits the graph structure *globally* in similarity computation as well.

In the literature, Ada has been applied to the tasks of link prediction [8, 19, 37] and recommendation [2] where it shows promising accuracy [8, 19]. In order to evaluate the effectiveness of Ada in the task of similarity computation of nodes in graphs, as our preliminary experiment, we applied it to Amazon [38] and DBLP [9] datasets, and compared its accuracy with those of SimRank and RWR[1]. The results are illustrated in Figure 1; as observed in the figure, Ada considerably outperforms *both* SimRank and RWR with the DBLP dataset and is somehow *comparable* to them with the Amazon dataset. Note that Ada is *not* a recursive similarity measure and considers only the direct neighbors of nodes a and b to compute $S(a, b)$ in Equation (2), while both SimRank and RWR recursively consider the graph structure globally to compute $S(a, b)$; thus, the observed results in Figure 1 are *quite promising*.

In this paper, we propose, *AdaSim*, a *recursive* similarity measure based on the *Ada philosophy*. The encouraging results in Figure 1 along with the fact that SimRank is a recursive generalization of the co-citation measure [31] that shows exploiting the global graph structure improves the co-citation accuracy significantly [13], motivated us to propose AdaSim hoping to improve the effectiveness of Ada in similarity computation of nodes. In AdaSim, the similarity scores of any node-pairs having common *direct* in-neighbors are computed based on Ada and these scores backwardly propagate in the graph to compute the similarity scores of node-pairs also based on their common *indirect* in-neighbors. In similarity computation, AdaSim basically employs *both* Ada and the pairwise normalization; this makes AdaSim be *robust* against a counter-intuitive property of the pairwise normalization [6, 9]. AdaSim shows *identical* accuracy to that of Ada on the first iteration and it is applicable to both directed and undirected graphs. Furthermore, we propose a *matrix form* for AdaSim to accelerate its computation; our matrix form provides the *exact* AdaSim scores and it is *dramatically faster* than the AdaSim iterative form in computation while provides an *identical* accuracy to that of the iterative form. Also, AdaSim scores *are* symmetric, bounded, monotonic, unique, and always existent.

We conduct extensive experiments with *five* real-world datasets to evaluate both the effectiveness and efficiency of our AdaSim in comparison with those of the existing similarity measures and also graph embedding methods (in short, embedding methods). The latter ones are widely applied to machine learning tasks (e.g., link prediction [5, 8, 32, 36] and node classification [5, 17, 26, 34]); inspired by the strategy observed in the word analogy detection [24, 34], we exploit the corresponding low-dimension vectors of nodes a and b to compute $S(a, b)$. Our experimental results demonstrate that 1) AdaSim *significantly* improves the accuracy of Ada, 2) it *outperforms* our competitors in terms of accuracy with *all* datasets, 3) its performance is *comparable* to that of SimRank* [40] while being better than other competitors, 4) AdaSim is *not* sensitive to the parameter tuning, and 5) embedding methods are *not* beneficial in similarity computation of nodes than similarity measures.

Our contributions in this paper are as follows:

- We propose AdaSim, a novel *recursive* similarity measure, which is based on the philosophy of Adamic/Adar in similarity computation.
- We provide a formula, which is *identical* to the original Adamic/ Adar on the first iteration and is applicable to both undirected and directed graphs.
- We propose a *matrix form* for AdaSim that is dramatically faster than its iterative form, while providing the exact similarity scores.
- We conduct extensive experiments with *five* real-world datasets to validate the effectiveness and efficiency of our AdaSim in comparison with existing methods.

The rest of this paper is organized as follows. In Section 2, we briefly discuss related work. In Section 3, we present our proposed measure, AdaSim, in detail. Section 4 explains the experimental settings and analyzes the results of our experiments. In Section 5, we conclude the paper.

2 RELATED WORK

In this section, we briefly explain the related work summarized in two categories of link-based similarity measures and graph embedding methods.

2.1 Link-based Similarity Measures

To compute the similarity score of node-pair (a, b), SimRank [13] employs a recursive philosophy; two random walkers, starting from a and b, traverse the graph recursively backward till they both arrive at a common node via identical path lengths from a and b. ASCOS++ [3] and SimRank* [40], two excellent variants of SimRank, exploit more paths in similarity computation than SimRank does, thereby regarding more number of node-pairs similar. CrashSim [18] is another nice variant of SimRank that enables similarity computation with provable approximation guarantees on temporal graphs. PSimRank [6], C-Rank [39], and JacSim [9] employ Jaccard coefficient [23] to address the pairwise normalization problem in SimRank; a more number of common neighbors may adversely affect the similarity score of a pair of nodes in comparison with another pair of nodes with a less number of common neighbors.

[1]Decay factor C and restart probability c are set as 0.6 and 0.15 for SimRank and RWR, respectively; number of iterations in both cases are 10.

PSimRank and C-Rank behave closely but they apply different values to normalize the similarity scores. JacSim avoids redundancy in computation and assigns an importance factor to the scores computed based on Jaccard and the pairwise normalization. MatchSim [20] applies a different solution to the same problem; instead of considering all pairwise neighbors of two nodes, it only exploits pairwise similar (matched) neighbors. P-Rank [42] and JPRank [10] exploit both in/out-neighbors in similarity computation, while the latter one also solves the pairwise normalization problem.

BEAR [15], a powerful variant of RWR [35], applies a block elimination approach to a preprocessed adjacency matrix for static graphs; for dynamic graphs, it updates the preprocessed matrix by assuming that only small parts of the matrix have been changed. BePI [14], another excellent variant of RWR, also utilizes a block elimination approach for preprocessing to achieve fast query time and incorporates an iterative method within the block elimination to optimize memory requirements.

2.2 Graph Embedding Methods

Embedding methods encode nodes with their connections in the graph into vectors in a continuous low-dimensional space where each dimension can be interpreted as a latent feature [5, 8, 26, 32]; they are applicable to a variety of machine learning tasks such as the link prediction [5, 8, 32, 36], node classification [5, 17, 26, 34], and word analogy detection [24, 34]. Similar to the strategy observed in the latter task, to compute the similarity score of a pair of nodes in a graph, we can apply Cosine [23] to their corresponding vector representations. DeepWalk [26], motivated by the achievement [24] in the representation learning for natural languages, considers a graph as an ordered sequence of nodes and tries to learn a model that maps neighbor nodes closer in the vector space. node2vec [8] considers the community of a node and its structural roles in the graph to capture the node's neighborhood by utilizing a biased random walk. NetMF [29] constructs a low-rank connectivity matrix for DeepWalk, which is explicitly factorized to obtain the representation vectors. DWNS [5] applies the adversarial training method [33] to DeepWalk in order to improve the robustness and generalization ability of the learning process.

3 PROPOSED MEASURE

Suppose that $G=(V,E)$ is a directed[2], homogeneous (i.e., nodes and links have a single type), and unsigned graph (i.e., links have only positive sign) where V represents a set of nodes, $E \in V \times V$ does a set of links among nodes, and I_a is a set of nodes directly pointing to a (i.e., direct in-neighbors of a). To apply Ada to directed graphs, we substitute N_a with I_a in Equation (2) as follows:

$$S(a,b) = \sum_{i \in I_a \cap I_b} \frac{1}{\log(|I_i| + e)} \tag{3}$$

where $w_i = \frac{1}{\log(|I_i|+e)}$ denotes the *weight* of node i; we add mathematical constant e to prevent the division by zero when $|I_i| \in \{0,1\}$.

As explained in Section 1, to compute the similarity score of a pair of nodes, Ada exploits *only* direct in-neighbors of the two

[2]In this paper, for the sake of *generality*, we focus on directed graphs since an undirected graph can be regarded as a directed one by considering two links in both directions instead of each single link.

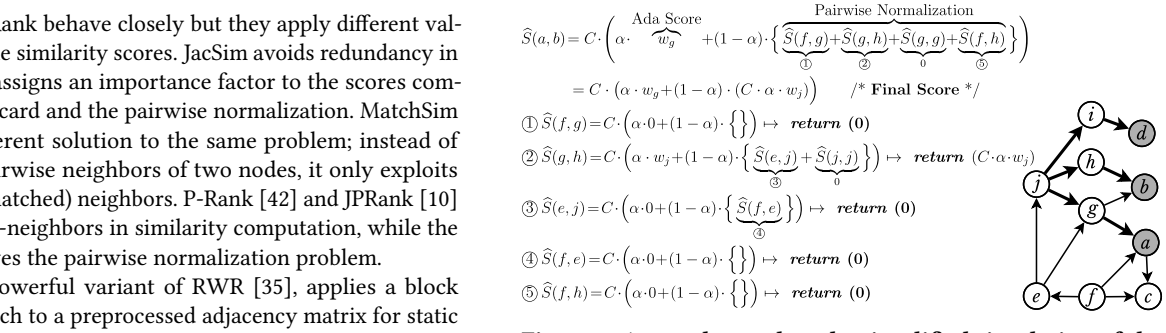


Figure 2: A sample graph and a simplified simulation of the AdaSim recursive computation.

nodes. Consider the sample graph in Figure 2 and the two node-pairs (a,b) and (b,d). By applying Equation (3), $S(a,b) = w_g = 0.64$ since $I_a \cap I_b = \{g\}$, while $S(b,d) = 0$ since $I_b \cap I_d = \emptyset$. However, node j indirectly points to d via node i and to b via nodes h and g, which means b and d have a common *indirect* in-neighbor as j; it implies that b and d can be somehow similar.

3.1 Intuitive Observation

Now, let us provide an *intuitive* algorithm to compute the similarity score of node-pair (a,b) in which we consider the *global* graph structure by exploiting *both* common direct and indirect in-neighbors of a and b based on the Ada philosophy (i.e., similar nodes have common in-neighbors; given that, they are regarded more similar if they have more common in-neighbors with fewer links in the graph) as follows. First, we sum up the weights of all nodes commonly connected to both a and b via paths with length *one* (i.e., common direct in-neighbors); next, we do the same calculation for the nodes commonly connected to both a and b via non-crossing paths with length *two* (i.e., common in-neighbors in two steps); we continue this process for the common in-neighbors in k steps. Finally, the summation of *all* these scores is regarded as the similarity score of (a,b). We formulate our intuitive algorithm as follows:

$$S_k(a,b) = \sum_{l=1}^{k} \sum_{i \in I_a^l \cap I_b^l} \sum_{p_a \in \{i \rightsquigarrow a\}^l} \sum_{p_b \in \{i \rightsquigarrow b\}^l} f(p_a, p_b).w_i \tag{4}$$

where I_a^l is a set of nodes connected to a via paths with length l $(1 \le l \le k)$, $\{i \rightsquigarrow a\}^l$ is a set of paths with length l from i to a, p_a is a single path $i \rightarrow r_1 \rightarrow \ldots \rightarrow r_{l-1} \rightarrow a$ ($\rightarrow$ denotes a direct path), and $f(p_a, p_b)$ is an indicator function that returns 0 if p_a and p_b cross at any nodes than i; otherwise 1 (i.e., we accumulate similarity scores by considering only the non-crossing independent paths).

To check our algorithm, let us consider the sample graph in Figure 2 and $k=2$. For node-pair (b,d), $I_b^1 \cap I_d^1 = \emptyset$ and $I_b^2 \cap I_d^2 = \{j\}$ (i.e., j connects to b via $j \rightarrow h \rightarrow b$ and $j \rightarrow g \rightarrow b$, and to d via $j \rightarrow i \rightarrow d$) where $f(j \rightarrow h \rightarrow b,\ j \rightarrow i \rightarrow d) = 1$ and $f(j \rightarrow g \rightarrow b,\ j \rightarrow i \rightarrow d) = 1$; therefore, $S(b,d) = w_j + w_j = 1.52$. For node-pair (a,b): $I_a^1 \cap I_b^1 = \{g\}$ and $I_a^2 \cap I_b^2 = \{e,j\}$ (i.e., e connects to each of a and b via one path and j connects to a and b via one and two paths, respectively) where for common in-neighbor e, $f(e \rightarrow g \rightarrow a,\ e \rightarrow g \rightarrow b) = 0$ (i.e., e does not contribute in similarity computation since paths from e to a and b cross each other at g); for common in-neighbor j,

$f(j\to g\to a,\ j\to g\to b)=0$ and $f(j\to g\to a,\ j\to h\to b)=1$; therefore, $S(a,b)=w_g+w_j=1.40$. Contrary to Ada, our intuitive algorithm considers b and d similar and also assigns a *higher* similarity score to a and b than the one Ada does by considering their common indirect in-neighbor j.

Equation (4) is tedious and suffers from high complexity if calculated directly. We propose a recursive formula for our intuitive algorithm represented by Equation (5) below with $S(a,a)=w_a$ as its base case, which is more straightforward and easier to compute:

$$S(a,b)=\sum_{i\in I_a}\sum_{j\in I_b}S(i,j) \tag{5}$$

Theorem 1 *For any node-pair (a,b), the calculated similarity score by Equation (5) is identical to that of Equation (4).*

Proof (By the mathematical induction) let us rewrite Equation (5) as the following iteration to a fixed-point, which is started by $S_0(a,b)=w_a$ if $a=b$; $S_0(a,b)=0$ otherwise:

$$S_{k+1}(a,b)=\begin{cases} w_a, & a=b \\ \sum_{i\in I_a}\sum_{j\in I_b}S_k(i,j), & a\neq b \end{cases} \tag{6}$$

in Equation (4), $S_1(a,b)=\sum_{i\in I_a\cap I_b}w_i$ since each of $\{i\rightsquigarrow a\}^1$ and $\{i\rightsquigarrow b\}^1$ has only one path as $i\to a$ and $i\to b$, respectively, which do not cross. In Equation (6), when $k=0$, also $S_1(a,b)=\sum_{i\in I_a\cap I_b}w_i$ since $S_0(i,j)=0$ for any node-pair (i,j) where $i\neq j$; thus, Equation (6) holds for $k=0$. We assume that the equation holds for k and rewrite Equation (4) by renaming the nodes as follows:

$$S_k(i,j)=\sum_{l=1}^{k}\sum_{r\in I_i^l\cap I_j^l}\ \sum_{p_i\in\{r\rightsquigarrow i\}^l}\ \sum_{p_j\in\{r\rightsquigarrow j\}^l}f(p_i,p_j).w_r \tag{7}$$

we rewrite Equation (6) as follows by replacing its right side with Equation (7):

$$\begin{aligned} S_{k+1}(a,b)&=\sum_{i\in I_a}\sum_{j\in I_b}S_k(i,j)\\ &=\sum_{i\in I_a}\sum_{j\in I_b}\Big\{\sum_{l=1}^{k}\sum_{r\in I_i^l\cap I_j^l}\ \sum_{p_i\in\{r\rightsquigarrow i\}^l}\ \sum_{p_j\in\{r\rightsquigarrow j\}^l}f(p_i,p_j).w_r\Big\}\\ &=\sum_{l=1}^{k+1}\sum_{r\in I_a^l\cap I_b^l}\ \sum_{p_a\in\{r\rightsquigarrow a\}^l}\ \sum_{p_b\in\{r\rightsquigarrow b\}^l}f(p_a,p_b).w_r \end{aligned} \tag{8}$$

since $i\in I_a$ and $j\in I_b$, there is a direct path from i and j to a and b, respectively, thus any nodes r commonly connected to i and j in k steps are also connected to a and b in $k+1$ steps.

Although Equation (5) provides identical similarity scores to the ones calculated by Equation (4) while being easier to implement and compute than that, the obtained similarity scores will not converge after continuous iterations.

Lemma 1 *The similarity scores obtained by Equation (5) are not convergence.*

Proof It is clear that the similarity scores obtained by Equation (5) are *monotonic* (i.e., $S_{k+1}(a,b)\geq S_k(a,b)$ for any k) and *not* bounded, which makes the convergence of similarity scores to finite values *not* guaranteed specifically in the case of undirected graphs. In the case of directed graphs, the scores may finally converge since empty in-neighbor sets result in stopping the recursion.

3.2 AdaSim: Iterative Form

Now, we are ready to propose a similarity measure that *preserves* the Ada philosophy in similarity computation, its accuracy on the first iteration being *identical* to that of original Ada, and provides similarity scores that *are* convergence to finite values. We present *AdaSim* that is initially based on Equation (5), while employing *both* Ada and the pairwise normalization in similarity computation 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 $S(a,b)=\widehat{S}(a,b)$, which is computed by the following recursive formula:

$$\widehat{S}(a,b)=C\cdot\Big(\frac{\alpha}{m}\sum_{i\in I_a\cap I_b}w_i+\frac{(1-\alpha)}{\sum_{r\in I_a}w_r\sum_{t\in I_b}w_t}\sum_{i\in I_a}\sum_{j\in I_b}w_i\cdot\widehat{S}(i,j)\cdot w_j\Big) \tag{9}$$

where $C\in(0,1)$ is a damping factor, $\alpha\in(0,1]$ is an *importance factor* to control the degree of importance of the score computed by Ada and the one computed by the pairwise normalization, and m is the *maximum* Ada score. $\widehat{S}(a,b)=0$ if $a=b$; it is the *base case* of the recursion where the result (i.e., 0) is returned without making another recursive call. To normalize the Ada score, we utilize the min-max normalization by dividing it to m. Also, in the pairwise normalization part, the total score is divided by the multiplication of the summation of weights for in-neighbors of a and b.

Let us consider our AdaSim in terms of a random walk model. We consider two random walkers who traverse the graph via in-links by starting from a and b *recursively backward* where the two nodes are regarded similar if random walkers meet up at a *common* direct or indirect in-neighbor of a and b; however, contrary to SimRank, their walking strategies are *not* completely independent as follows. These two random walkers are supposed to meet up with the highest probability (i.e., 1) at the common nodes between I_a and I_b (if existed) on the first step (i.e., the similarity is computed by Ada) or they traverse the graph independently to meet up at common indirect in-neighbors of a and b (i.e., the similarity is computed by the pairwise normalization). Furthermore, SimRank assumes that all in-neighbors of a are *equally significant* to be visited by a random walker through assigning an *identical* probability value as $\frac{1}{|I_a|}$ to all a's in-links; instead, AdaSim regards that in-neighbors with *higher* weights are *more* likely to be visited by the random walker through assigning a *different* probability value as $\frac{w_i}{\sum_{r\in I_a}w_r}$ to the in-link from each of in-neighbors i (i.e., $i\in I_a$).

In Figure 2, we illustrate a *simplified* simulation of the AdaSim recursive computation for node-pair (a,b) in our sample graph; to simplify the figure, we do not normalize the scores. Since $a\neq b$, $S(a,b)$ is computed by $\widehat{S}(a,b)$ as follows. The similarity score based on common direct in-neighbors (i.e., g) is computed by Ada and five recursive calls (i.e., indicated in order by circled numbers) are executed to exploit the common indirect in-neighbors (i.e., j) of a and b in similarity computation; each recursive call returns its computation result to its parent. The similarity score between g and h (i.e., having common direct in-neighbors j) is computed based on Ada and this score backwardly propagates in the graph to complete the similarity computation for (a,b). Note that *without* applying parameters C, α, and any normalization to AdaSim, it is *identical* to our intuitive algorithm represented in Section 3.1.

The recursive computation to obtain $S(a,b)$ can be solved by the iteration to a fixed-point for $k=1,2,\ldots$ over $\widehat{S}(a,b)$ as follows: if

Algorithm 1: AdaSim Iterative Form Computation

Input : directed graph $G(V,E)$, damping factor C, importance parameter α, number of iterations k
Output: AdaSim scores $S(*,*)$

1 initialize $[S]_{*,*}$; $[\widehat{S}]_{*,*}$; $[M]_{*,*}$; $[\widehat{\omega}]_*$; m; extract I_a for all nods a
2 **for** $a \leftarrow 1$ **to** V **do**
3 **for** $b \leftarrow 1$ **to** V **do**
4 **if** $a \neq b$ **and** $I_a, I_b \neq \emptyset$ **then**
5 $common$=Intersection(I_a, I_b); sum=0
6 **for** $i \in common$ **do** sum+=$1/\log(e+|I_i|)$
7 $[M]_{a,b}$=sum
8 **if** $sum > m$ **then** m=sum
9 **for** $i \in I_a$ **do** $[\widehat{\omega}]_a$+=$1/\log(e+|I_i|)$
10 M=M/m; S=$C \cdot \alpha \cdot M$
11 **for** $itr \leftarrow 2$ **to** k **do**
12 $\widehat{S} \leftarrow S$
13 **for** $a \leftarrow 1$ **to** V **do**
14 **for** $b \leftarrow 1$ **to** V **do**
15 **if** $a \neq b$ **and** $I_a, I_b \neq \emptyset$ **then**
16 sum=0
17 **for** $i \in I_a$ **do**
18 **for** $j \in I_b$ **do**
19 sum+=$\frac{[\widehat{S}]_{i,j}}{\log(e+|I_i|)\cdot\log(e+|I_j|)}$
20 **if** $[\widehat{\omega}]_a \neq 0$ ***and*** $[\widehat{\omega}]_b \neq 0$ **then**
21 $[S]_{a,b}$=$C\cdot(\alpha\cdot[M]_{a,b}+(1-\alpha)\cdot sum/([\widehat{\omega}]_a\cdot[\widehat{\omega}]_b))$
22 **else** $[S]_{a,b}$=$C\cdot\alpha\cdot[M]_{a,b}$
23 setDiag(S,1)
24 **return** $S(*,*)$

$a=b$, $S_k(a,b)=1$ for *any* k; if $a \neq b$, $S_k(a,b)$ is computed by $\widehat{S}_k(a,b)$. In the iterative computation, $\widehat{S}_k(a,b) = 0$ if $a = b$; otherwise:

$$\widehat{S}_k(a,b) = C \cdot \Big(\frac{\alpha}{m} \sum_{i \in I_a \cap I_b} w_i + \frac{(1-\alpha)}{\Sigma_{r \in I_a} w_r \Sigma_{t \in I_b} w_t} \sum_{i \in I_a} \sum_{j \in I_b} w_i \cdot \widehat{S}_{k-1}(i,j) \cdot w_j\Big) \quad (10)$$

where the iterative computation starts with $\widehat{S}_0(a,b)=0$ for all node-pairs (a,b).

It is worth to note the following points regarding the AdaSim iterative form. $\widehat{S}_1(a,b)$ is computed *only* based on the Ada score since $\widehat{S}_0(i,j) = 0$ for all node-pairs (i,j), which means $S_1(a,b) = \widehat{S}_1(a,b) = \frac{C\cdot\alpha}{m} \cdot \Sigma_{I_a \cap I_b} w_i$. Also, on the first iteration, AdaSim provides an *identical* accuracy to that of Ada in similarity computation since the original Ada scores are universally multiplied by the constant $\frac{C\cdot\alpha}{m}$. The AdaSim scores are symmetric, bounded, monotonic, unique, and always existent as shown in Appendix A[3].

3.2.1 Algorithm and Computation. Algorithm 1 represents the computation of the AdaSim iterative form. In line 1, entries of matrices S, $\widehat{S}$, M (all of them with size $|V|\times|V|$), and vector $\widehat{\omega}_{|V|\times 1}$ along with variable m are initialized by zero and I_a is extracted for all nodes a. In lines 2 to 9, Ada scores are calculated and stored in M, m is identified, and entries of $\widehat{\omega}$ are set to the summation of in-neighbors weights for nodes. In line 10, Ada scores are normalized and AdaSim scores are calculated on the first iteration. In lines 11 to 22, AdaSim scores are iteratively computed. In line 23, the diagonal values of S are set to 1. The space complexity is $O(|V|^2)$ and the time complexity is $O(|V|^4)$ in the worst case.

[3]The Appendix is accessible via https://github.com/mrhhyu/AdaSim

3.3 AdaSim: Matrix Form

In this section, we propose a matrix form for AdaSim, which *not* only accelerates the computation of our iterative form but also provides the *exact* AdaSim scores. We start by providing a matrix form for original Ada in Equation (3). Instead of considering only common direct in-neighbors of a and b to compute their Ada score, we can consider *all* the nodes in the graph, thus:

$$Ada(a,b) = \sum_{i \in I_a \cap I_b} w_i = \sum_{i \in V} [A]_{i,a} \cdot w_i \cdot [A]_{i,b} \quad (11)$$

where $A_{|V|\times|V|}$ is the adjacency matrix of graph G. If $[A]_{i,a} = [A]_{i,b} = 1$, means $i \in I_a \cap I_b$ (i.e., there is an in-link from i to both a and b); otherwise, $i \notin I_a \cap I_b$. $Ada(a,b)$ in Equation (11), is identical to the entry $[M]_{a,b}$ of the following matrix:

$$M = \big((A \odot \omega)^T \cdot A\big) \wedge I_0 = (W^T \cdot A) \wedge I_0 \quad (12)$$

where ω is a *column* vector (i.e., with size $|V|\times 1$) containing the weights of nodes in the graph (i.e., $[\omega]_i = w_i = \frac{1}{log(|I_a|+e)}$) and $\odot$ denotes the *Hadamard product* [12] as $[A \odot \omega]_{i,j} = [A]_{i,j} \cdot [\omega]_i$. We define $W = A \odot \omega$, $W \in \mathbb{R}^{|V|\times|V|}$, as a *weight matrix* where $[W]_{i,j} = w_i$ if $[A]_{i,j} \neq 0$ (i.e., $i \in I_j$), and entries in its column j store the weights for in-neighbors of node j. W^T is the transpose of W, $\wedge$ is the *conjunction operator* selecting the minimum operand (i.e., $(X \wedge Y) = U$, then $[U]_{a,b} = \min\{[X]_{a,b}, [Y]_{a,b}\}$), and in matrix $I_{0\,|V|\times|V|}$, diagonal entries are 0 and others are 1.

Now, we rewrite the iterative form in Equation (9) as follows:

$$\widehat{S}(a,b) = C \cdot \Big(\frac{\alpha \cdot [M]_{a,b}}{\max(M)} + (1-\alpha) \sum_{i \in I_a} \sum_{j \in I_b} \frac{w_i}{\Sigma_{r \in I_a} w_r} \cdot \widehat{S}(i,j) \cdot \frac{w_j}{\Sigma_{t \in I_b} w_t}\Big) \quad (13)$$

where $\max(M)$ denotes the maximum value in M.

Similar to Equation (11), we again consider all the nodes in the graph instead of only ones in I_a and I_b, thus:

$$\widehat{S}(a,b) = C \cdot \Big(\frac{\alpha \cdot [M]_{a,b}}{\max(M)} + (1-\alpha) \cdot \sum_{i \in V} \sum_{j \in V} \Big(\frac{[W]_{i,a}}{\Sigma_{r \in V} [W]_{r,a}}\Big) \cdot \widehat{S}(i,j) \cdot \Big(\frac{[W]_{j,b}}{\Sigma_{t \in V} [W]_{t,b}}\Big)\Big) \quad (14)$$

where, as already explained, if $[W]_{i,a} \neq 0$, indicates that node i is directly connected to a (i.e., $i \in I_a$).

Finally, we provide the following recursive matrix form for our AdaSim:

$$\begin{cases} \widehat{S} = C \cdot \Big(\frac{\alpha \cdot M}{\max(M)} + (1-\alpha) \cdot \overline{W}^T \cdot \widehat{S} \cdot \overline{W}\Big) \wedge I_0 \\ S = \widehat{S} \vee I \end{cases} \quad (15)$$

where $S \in \mathbb{R}^{|V|\times|V|}$ denotes a similarity matrix whose entry $[S]_{a,b}$ contains the AdaSim score of node-pair (a,b), $\overline{W}$ is the *column normalized* weight matrix whose entry $[\overline{W}]_{i,j} = \frac{w_i}{\Sigma_{r \in V} [W]_{r,j}}$, $\vee$ is the *disjunction operator* selecting the maximum operand (i.e., $(X \wedge Y) = U$, then $[U]_{a,b} = \max\{[X]_{a,b}, [Y]_{a,b}\}$), and $I_{|V|\times|V|}$ is an identity matrix. By applying $\wedge$ and $\vee$ operators, we set diagonal entries in $\widehat{S}$ and S as zero and one, respectively, in accordance with our iterative form. The recursive matrix form in Equation (15) can

be written in the following iterative form for $k = 1, 2, ...$:

$$\begin{cases} \widehat{S}_k = C \cdot \left(\frac{\alpha \cdot M}{\max(M)} + (1-\alpha) \cdot \overline{W}^T \cdot \widehat{S}_{k-1} \cdot \overline{W}\right) \wedge I_0 \\ S_k = \widehat{S}_k \vee I \end{cases} \tag{16}$$

where the iterative computation is started with $\widehat{S}_0 = Z$, a zero matrix with 0 in all entries.

We clarify the following points regarding our matrix form. The straightforward mathematical process employed to transform the AdaSim iterative form into the matrix form clearly shows that our matrix form provides *exact* AdaSim scores *without* requiring any approximation. The space complexity is $O(|V|^2)$ likewise the iterative form. We have a Hadamard product and a matrix multiplication on the first iteration, and two matrix multiplications on subsequent iterations; both matrices A and W are represented by a *compressed sparse column* (CSC) storage schema [30]. The matrix multiplications are more time-consuming than Hadamard; let d be the number of non-zero entries in W. By employing the CSC representation, the time complexity of our matrix form is $O(d|V|)$, which is quite faster than the AdaSim iterative form as shown in Section 4.2.1.

3.4 Discussions

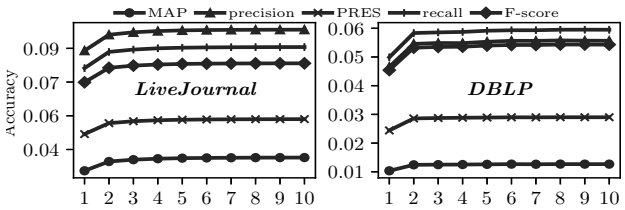
In this section, we provide some discussions regarding our similarity measure, AdaSim.

Pure pairwise normalization (PNN): one may suggest that applying the *only* pairwise normalization to Equation (5) is enough to recursively compute the convergent similarity scores. Under this assumption, we consider the following iterative form where $S_0(a,b) = w_a$ if $a = b$; otherwise $S_0(a,b) = 0$:

$$S_k(a,b) = \underbrace{\frac{C}{\Sigma_{r \in I_a} w_r \Sigma_{t \in I_b} w_t}}_{①} \cdot \overbrace{\sum_{i \in I_a} \sum_{j \in I_b} w_i \cdot S_{k-1}(i,j) \cdot w_j}^{②} \tag{17}$$

We conducted an experiment by employing the above formula (i.e., PPN) for similarity computation in ten iterations, and compared its accuracy with that of Ada. Figure 3 illustrates the results with LiveJournal [38] and DBLP datasets; the accuracy of Ada is shown in a table under the plot. Although PPN computes the similarity scores recursively, its accuracy is significantly *less* than that of Ada with both datasets on *all* iterations. The reason is that Equation (17) *suffers* from the pairwise normalization problem. Let's consider a simple example as follows. Assume that nodes a and b have a *larger* number of in-neighbor pairs than that of a and c, however, the common direct in-neighbors (and consequently, their weights summation) between a and b are same with that of a and c; it means a is similar to b as much as it is similar to c. However, in Equation (17), the summation of the common in-neighbors weights (i.e., part ②) is divided by considering the weights of *all* pairs of in-neighbors (i.e., part ①[4]), thereby *adversely* affecting the similarity score between a and b, resulting in $S_1(a,b) < S_1(a,c)$. These scores propagate in the graph and adversely affect the similarity scores computed on the subsequent iterations. Our AdaSim is *robust* against this problem by applying the min-max normalization to the similarity scores on the first iteration. More specifically, the

[4]Note that $\Sigma_{r \in I_a} w_r \cdot \Sigma_{t \in I_b} w_t = \Sigma_{r \in I_a} \Sigma_{t \in I_b} w_r \cdot w_t$



Ada	MAP	precision	PRES	recall	F-score
LiveJournal	0.063	0.140	0.082	0.118	0.111
DBLP	0.090	0.185	0.114	0.153	0.154

Figure 3: Accuracy comparison of Ada and PPN.

pairwise normalization may change the node-ranking via dividing each of similarity scores by a *different* value, while the min-max normalization divides *all* similarity scores by an *identical* value, thereby *not* changing the node-ranking.

Min-max normalization: *instead* of m, we can also utilize $|V|$ to normalize Ada scores. In the Ada philosophy, *not* only the number of common neighbors but *also* their weights are important in similarity computation; thus, let us consider the two special cases leading to a large similarity score for a node-pair (a,b) in a given graph G (for simplicity, we suppose that G is undirected). In the first case, a and b are connected to a large number of nodes with large weight values; in the second case, they are connected to a large number of nodes with small weight values as follows: 1) $V = \{a, b, v_i | 0 \le i \le |V| - 2\}$, a and b are commonly connected to nodes v_i (i.e., $N_a \cap N_b = \{v_i\}$), and there are no links between any of nodes v_i. Thus, $S_1(a, v_i) = S_1(b, v_i) = 0$, $S_1(v_i, v_j) = w_a + w_b = \frac{2}{\log(|V|-2+e)}$ (i.e., $i \neq j$), and $S_1(a,b) = \Sigma_i w_{v_i} = \frac{|V|-2}{\log(2+e)}$; therefore, if $|V| \to \infty$, $0 \le S_1(*,*) < |V|$. 2) G is a complete graph, then $S_1(a,b) = \frac{|V|-2}{\log(|V|-1+e)}$; therefor, if $|V| \to \infty$, $0 \le S_1(*,*) < |V|$. Note that using $|V|$ instead of m, can *improve* the performance of our matrix form (i.e., we do not need to find m) not that of the iterative form since as shown in Algorithm 1, m is identified while Ada scores are calculated. Although for a large value of $|V|$, the normalized similarity scores become very small, it does *not* affect the node-ranking since all scores are divided by an identical value.

Applicability to undirected graphs: although for generalization, we discussed our AdaSim w.r.t directed graphs, the both iterative and matrix forms are applicable to directed as well as undirected graphs. In the case of undirected graphs, it is sufficient that two links in both directions are regarded for each single link.

4 EXPERIMENTAL EVALUATION

In this section, we evaluate the effectiveness (i.e., accuracy) and efficiency (i.e., performance) of AdaSim in comparison with those of other measures.

4.1 Experimental Settings

We employ following five *real-world* datasets summarized in Table 1: 1) *Amazon* [38] is a graph of co-purchasing products in the Amazon website. The node labels denote the product category; to perform reasonable evaluation, we neglected labels with less than ten nodes. It is partially tagged by 71 different labels. 2) *DBLP* [9] is a citation graph made of papers in the areas of data mining and databases published in 2006 and earlier. The node labels indicate the papers'

Table 1: Some statistics about our datasets

	$\|V\|$	$\|E\|$	#Labels	Graph Type
Amazon	30,623	97,478	71	undirected
LiveJournal	11,755	160,046	7,086	undirected
DBLP	21,177	248,131	11	directed
TREC	43,202	347,702	16	directed
Wikipedia	4,777	184,812	40	undirected

research topics; node labeling is done based on a famous data mining textbook [11] where the papers relevant to a research topic are addressed in the bibliographic section of chapters. The graph is partially tagged by 11 different labels corresponding to 11 chapters. 3) *LiveJournal* [38] is a graph representing social relations among bloggers and the node labels denote bloggers interests. To perform reasonable evaluation, we chose only labels with more than ten and less than a hundred nodes. It is partially tagged by 7,086 different labels. 4) TREC [9] is a graph of webpages constructed based on TREC 2003[5]. The node labels indicate the relevant query topic for the webpages created based on LETOR 3.0 [27], released by Microsoft Research Asia. It is partially tagged by 16 different labels. 5) *Wikipedia* [8, 36] 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 different labels.

We evaluate our AdaSim (`AS`) in comparison with eight popular as well as the state-of-the-art methods: Ada [1], SimRank (`SR`) [13], SimRank* (`SR*`) [40], JacSim (`JS`) [9], DeepWalk (`DPW`) [26], DWNS (`DWN`) [5], NetMF (`Net`) [29], and node2vec (`n2v`)[6] [8] with *both* directed and undirected graphs. Since it has been shown that `SR*` outperforms ASCOS++ and produces same results with RWR based measures with undirected graphs while it outperforms them with directed ones [40], we do not consider those measures in our evaluation. With the same reason, we do not consider C-Rank [39], PSimRank [6], and MatchSim [20] since `JS` outperforms them [9]. Our preliminary experiments showed that `DPW`, `DWN`, `Net`, and `n2v` outperform GraphGAN [36], APT [32], BoostNE [17], and Line [34]; thus, we do not consider them in this evaluation. We implemented all the similarity measures by their *matrix forms* with the parameter settings suggested by their original work: damping factor C is set as 0.6 for all measures; for JacSim, importance factor α is set as 0.4 by following [9]; all measures are performed in ten iterations. For embedding methods, we used the implementations with the default parameter settings provided by their original work[7] and set the vector dimensions d as 128.

We utilize MAP, precision, recall, F-score [23], and PRES [22] as accuracy evaluation metrics. In each dataset, we consider the labels as ground truth sets and use *every* single node in a label l as a *query node for similarity based searching*. We find the top-t ($t = 5, 10, 20, 30$) nodes considered similar to the query; if a node belongs to l, it is regarded as *relevant*, otherwise *irrelevant*. For each value of t, after computing the AP (average precision) [23], precision, recall, F-score, and PRES for *all* the query nodes in l, we took their average values to get MAP and averages of precision, recall, F-score, and PRES for l itself. Then, we compute the average value of each metric over *all* labels as the corresponding accuracy for each t. Finally, the average value of each metric over all values of t, is regarded as the *final* accuracy with the dataset.

[5]http://trec.nist.gov/data.html
[6]In this section, these abbreviations are used in figures, tables, and explanations.
[7]DPW [25], DWN [4], Net [28], n2v [7]

Table 2: Execution times (minutes) of AdaSim matrix form and iterative form

	Amazon	DBLP	LiveJournal	TREC	Wikipedia
Iterative Form	168.80	476.78	728.16	4270.38	1008.30
Matrix Form	1.31	0.86	0.48	4.19	0.13

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.

4.2 Results and Analyses

4.2.1 Performance comparison of our matrix form and iterative form. As explained in Section 3.2, our matrix form provides exact AdaSim scores, while it accelerates the similarity computation. We evaluate its performance in comparison with that of the iterative form with our datasets only in five iterations. The results are shown in Table 2; the matrix form is *dramatically* faster than the iterative form with *all* datasets. Therefore, *hereafter, we employ the AdaSim matrix form in our experiments* in this section. It is worth to note that the time complexity of the matrix form is $O(|V|d)$,which mainly depends on the number of nodes in the graph; TREC, as our biggest dataset, has the longest execution time (i.e., 4.19), while Wikipedia, as the smallest dataset, has the shortest one (i.e., 0.13). However, the time complexity of the iterative form is $O(|V|^2\hbar^2)$ where $\hbar$ is the average number of neighbors per node; for example, although Wikipedia has a very few number of nodes (i.e., 4,777) than Amazon (i.e., 30,623), it shows the worst performance (i.e., 1008.30) since its value of $\hbar$ is 38.68, while this number is 3.43 for Amazon.

4.2.2 Parameter tuning. `AS` has two parameters: damping factor C and importance factor α. The results of our experiments with different values of C (as 0.4, 0.6, and 0.8) demonstrate that the accuracies of `AS` equipped with different values of C are *not* tangible with all datasets; we set the value of C as 0.6 for `AS`, in accordance to other measures listed in Section 4.1. In the case of α, we extensively investigate how the accuracy of `AS` in similarity computation changes with different values of α as follows. With each dataset, we set the value of α in the range [0.1, 0.9] in step of 0.1, and evaluate the accuracy of `AS` for each value of α in ten iterations (i.e., $45 = (9 \times 5)$ different cases). We do not consider 0 and 1 values; when $\alpha = 0$, the similarity scores will be zero on *all* iterations since the similarity scores on the first iteration are zero; when $\alpha = 1$, the similarity is computed based on *only* Ada scores on all iterations. Table 3 shows the results of parameter tuning for DBLP and LiveJournal datasets where the values in parentheses denote the iterations on which the best accuracy were observed; for example, when $\alpha = 0.1$ with the DBLP dataset, the best accuracy was observed on iteration 4.

With both datasets, `AS` shows better accuracy when α is set as 0.6, 0.7, and 0.8; it shows the best one when $\alpha = 0.7$. Also, we observed the same circumstances with other datasets; the best values of α are 0.8 for Amazon and 0.7 for TREC and Wikipedia (for the sake of

Table 3: The results of parameter tuning with DBLP and LiveJournal datasets

DBLP	0.1 (4)	0.2 (3)	0.3 (3)	0.4 (4)	0.5 (3)	0.6 (4)	0.7 (4)	0.8 (4)	0.9 (4)
MAP	0.1008	0.1012	0.1016	0.1019	0.1020	0.1022	**0.1022**	0.1019	0.1015
precision	0.2133	0.2139	0.2145	0.2146	0.2144	0.2150	**0.2151**	0.2150	0.2145
PRES	0.1305	0.1310	0.1312	0.1314	0.1315	0.1318	**0.1319**	0.1317	0.1313
recall	0.1805	0.1809	0.1810	0.1811	0.1807	0.1811	**0.1811**	0.1811	0.1802
F-score	0.1792	0.1797	0.1800	0.1800	0.1797	0.1802	**0.1802**	0.1802	0.1795
LiveJournal	0.1 (2)	0.2 (2)	0.3 (3)	0.4 (3)	0.5 (3)	0.6 (3)	0.7 (4)	0.8 (4)	0.9 (3)
MAP	0.0719	0.0722	0.0727	0.0730	0.0731	0.0733	**0.0733**	0.0731	0.0728
precision	0.1547	0.1554	0.1560	0.1567	0.1573	0.1578	**0.1581**	0.1581	0.1578
PRES	0.0927	0.0931	0.0935	0.0939	0.0942	0.0944	**0.0945**	0.0944	0.0941
recall	0.1312	0.1317	0.1321	0.1326	0.1331	0.1334	**0.1336**	0.1335	0.1332
F-score	0.1244	0.1249	0.1253	0.1258	0.1263	0.1267	**0.1268**	0.1268	0.1265

brevity, we set $\alpha = 0.7$ with *all* datasets in this section). This results imply that AS *is not that sensitive* to the values of α and C.

4.2.3 Accuracy comparison with undirected graphs. Now, we compare the accuracy of AS with those of Ada, SR, SR*, JS, DPW, DWN, Net, and n2v in similarity computation with Amazon, LiveJournal, and Wikipedia datasets in terms of MAP, precision, PRES, recall, and F-score. For similarity measures, we consider their *best* accuracy observed in ten iterations and represent it with notion "X−#", which means measure X on iteration #; for example, with the Amazon dataset, AS-7 denotes AS shows its best accuracy on iteration 7. Figure 4 illustrates the results; 1) with the Amazon dataset, Ada shows the *worst* accuracy among all methods. Although, the accuracy of other methods are somehow comparable to each other, AS and SR* show the best accuracy; however, AS outperforms SR* in terms of all metrics except MAP. Among embedding methods, Net shows the worst accuracy, while three other methods show close accuracies. 2) With the LiveJournal dataset, SR shows the worst accuracy, while our AS provides the *best* one in terms of MAP, precision, PRES, recall, and F-score. Among embedding methods, DPW outperforms DWN, Net, and n2v. 3) With the Wikipedia dataset, SR, SR*, DWN, and n2v show the lowest accuracy, while our AS *outperforms* all other methods in terms of all metrics.

Table 4 represents the *percentage of improvements* in accuracy obtained by AS over all other methods with our three undirected graphs. Our AS significantly *improves* the accuracy of Ada in similarity computation with *all* datasets; this improvement is *dramatically* noticeable with the Amazon dataset where accuracy in all metrics is improved more than 110%. The reason is that AS exploits the *global* graph structure in similarity computation, while Ada exploits the graph structure *locally*. Although the accuracy of SR* and JS are slightly close to those of AS with Amazon and Wikipedia datasets, respectively, our AS outperforms *all* methods with *all* datasets. These findings imply that the Ada philosophy is *beneficial* in similarity computation of nodes in graphs.

4.2.4 Accuracy comparison with directed graphs. Now, we compare the accuracy of our AS with those of other methods in similarity computation with DBLP and TREC datasets in terms of MAP, precision, PRES, recall, and F-score. Figure 5 illustrates the results; 1) with the DBLP dataset, among similarity measures, SR and SR* show the worst accuracy in terms of all five metrics, while among embedding methods, Net and n2v show the lowest accuracy and DPW shows the highest one; AS significantly *outperforms all* other

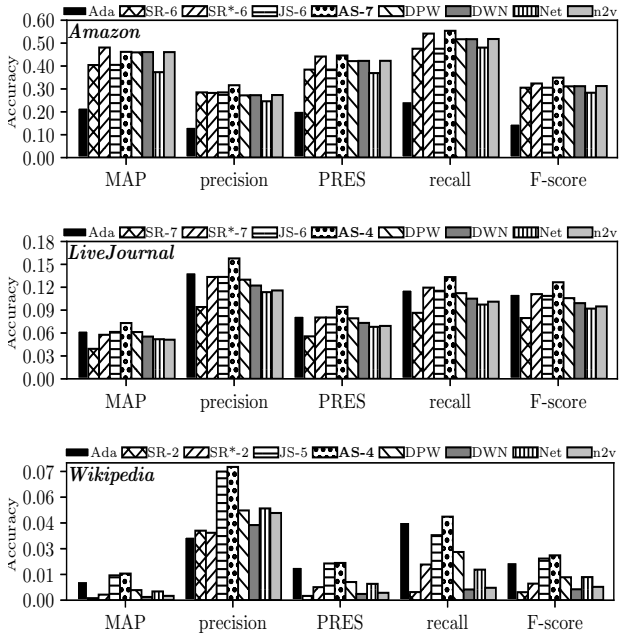


Figure 4: Accuracy comparison with undirected graphs.

Table 4: Accuracy improvements (%) by AS over other methods with undirected graphs

Amazon	Ada	SR	SR*	JS	DPW	DWN	Net	n2v
MAP	110.97	14.22	-3.94	14.03	0.48	0.19	23.75	0.22
precision	135.05	10.98	11.91	10.99	16.30	15.96	28.44	15.68
PRES	118.11	16.01	0.98	15.93	5.77	5.54	20.76	5.47
recall	124.59	16.45	2.24	16.46	7.12	7.06	15.27	6.92
F-score	134.66	14.38	7.89	14.41	12.22	11.99	23.32	11.76
LiveJournal	Ada	SR	SR*	JS	DPW	DWN	Net	n2v
MAP	15.46	86.03	26.76	18.99	19.26	32.08	40.91	42.66
precision	12.98	67.69	18.23	18.18	21.45	29.08	38.91	36.28
PRES	14.10	69.48	17.39	17.48	18.96	28.77	38.49	36.11
recall	13.92	54.17	11.55	15.54	18.75	26.88	37.00	31.87
F-score	13.63	58.90	14.01	16.48	19.67	27.60	37.56	33.40
Wikipedia	Ada	SR	SR*	JS	DPW	DWN	Net	n2v
MAP	39.01	1213.56	379.88	7.19	164.05	724.47	203.92	517.53
precision	109.64	91.73	97.73	3.60	48.49	77.36	45.24	52.81
PRES	11.75	774.60	185.77	1.40	105.01	497.52	128.08	404.42
recall	6.78	930.93	134.28	28.46	73.29	679.81	173.52	573.96
F-score	17.71	460.52	170.95	7.71	95.36	310.69	94.20	236.60

methods in terms of all metrics. 2) With the TREC dataset, among similarity measures, SR shows the worst accuracy, and among embedding methods, DWN and n2v show the worse accuracy, while DPW shows the best one. Although SR* shows slightly better accuracy than AS in terms of only precision and F-score, our AS significantly outperforms it in terms of MAP, PRES, and recall; AS shows the *best* accuracy among *all* methods. Table 5 represents the percentage of improvements in accuracy obtained by AS over all other methods with the two datasets. AS significantly *improves* the accuracy of Ada in similarity computation with the directed graphs as well, which is in accordance with our observation for undirected graphs.

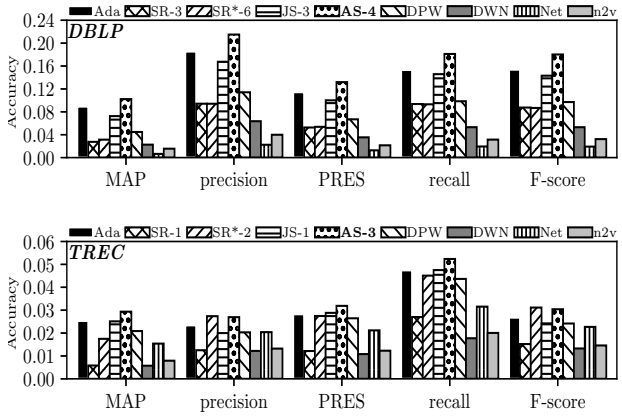


Figure 5: Accuracy comparison with directed graphs.

Table 5: Accuracy improvements (%) by AS over other methods with directed graphs

DBLP	Ada	SR	SR*	JS	DPW	DWN	Net	n2v
MAP	14.67	271.57	226.06	40.97	128.42	351.90	1482.35	556.10
precision	16.04	128.28	128.52	28.37	88.40	238.17	858.72	438.81
PRES	15.39	150.66	145.25	31.70	97.09	272.51	939.16	513.63
recall	18.25	93.54	94.46	24.20	83.90	240.80	831.58	476.02
F-score	17.24	105.96	107.62	25.87	85.61	239.24	837.41	457.63
TREC	Ada	SR	SR*	JS	DPW	DWN	Net	n2v
MAP	16.01	404.12	67.95	16.85	40.58	412.04	90.64	270.45
precision	15.33	115.52	-1.57	34.77	32.64	121.00	32.06	103.94
PRES	13.01	162.39	16.18	10.62	20.53	194.91	50.59	159.40
recall	10.49	93.93	16.12	10.21	19.95	195.32	66.01	161.15
F-score	13.73	100.13	-2.38	25.46	25.78	129.09	33.98	108.50

4.2.5 Performance comparison. In this paper, although we mainly focus on the accuracy than performance (i.e., time), to conduct a fair comparison, we implemented the *matrix forms* of SR, SR*, JS, and AS based on the CSC representation *without* applying any acceleration techniques such as fine-grained memorization [40], partial sums memoization [21], and Monte Carlo sampling [41]. With each dataset, we ran a similarity measure in ten iterations for five times and their *average* run time is regarded as its final execution time with that dataset. For embedding methods, we measure the execution time as the *summation* of a *learning time* (i.e., elapsed time to construct low-dimensional vectors) and a *vector-similarity computation time* (i.e., elapsed time to compute the similarity scores of *all* vector-pairs by employing Cosine). We also implemented Cosine based on a matrix/vector multiplication approach, which is significantly faster than its conventional implementation. In our comparison, we do *not* consider the required time to store the results of similarity computation in a file or a database. The vector-similarity computation times (in minutes) are 42.31, 23.25, 7.33, 74.80, and 1.23 for Amazon, DBLP, LiveJornal, TREC, and Wikipedia datasets, respectively. Table 6 represents the execution times of all methods with our five datasets.

SR* shows the best performance due to its excellent formula, which requires only one matrix multiplication; however, our AS shows *comparable* performance with SR*. AS shows slightly better performance than SR since on the first iteration, it has one matrix multiplication and a Hadamard product, while SR has two matrix multiplications; their performance on subsequent iterations are not tangible since they both have two matrix multiplications. Among similarity measures, JS has the worst performance since it employs one pairwise normalization on each iteration to compute matrix E [9, Equation (19)]. Among embedding methods, DPW and DWN show the worse performance (i.e., due to their long learning times), while that of n2v is the best one.

Table 6: Execution times (minutes) of all methods

	AS	SR	SR*	JS	DPW	DWN	Net	n2v
Amazon	2.98	3.71	2.81	6.83	136.98	204.56	48.74	44.39
DBLP	1.83	1.96	1.39	4.21	27.5	108.25	29.21	23.43
LiveJournal	0.86	0.94	0.68	5.17	61.62	41.53	8.17	8.54
TREC	9.12	9.38	6.74	37.42	138.14	480.03	85.63	75.89
Wikipeida	0.25	0.31	0.14	62.12	16.91	11.31	1.52	5.27

5 CONCLUSIONS

In this paper, we proposed AdaSim, a novel recursive similarity measure for graphs. AdaSim is designed based on the Ada philosophy; to compute the similarity score of a node-pair (a, b), AdaSim *not* only recursively considers the common in-neighbor nodes of a and b but *also* considers the weights of such common nodes in similarity computation. We conducted extensive experiments with five real-world datasets to evaluate both accuracy and performance of our AdaSim in comparison with those of popular as well as the state-of-the-art similarity measures and also embedding methods. Our experimental results demonstrate that 1) AdaSim significantly improves the accuracy of Ada with all datasets; 2) AdaSim outperforms all similarity measures and embedding methods in terms of accuracy with all datasets, thanks to the Ada philosophy; 3) with all datasets, its performance is comparable to that of SimRank*, while being better than other methods; 4) AdaSim is not that sensitive to the parameter tuning; 5) although embedding methods, mainly DeepWalk, show promising accuracy in some cases (e.g., with Amazon dataset), computing the similarity of vectors is tedious; also, in comparison with similarity measures, they have a large number of parameters, leading to a time-consuming parameter tuning.

We figured out interesting directions for our future work as follows. We plan to apply an acceleration technique such as fine-grained memorization [40] and partial sums memoization [21] to only the pairwise normalization part of the AdaSim computation since matrix M in Equation (16) is computed once and reused in all iterations. In addition, we plan to extend our measure to cover a special kind of graphs, *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 [16]; however, existing similarity measures consider only a single link type (i.e., positive).

ACKNOWLEDGMENTS

This work was supported by the National Research Foundation of Korea (NRF) grants funded by the Korea government (MSIT) (No. NRF-2020R1A2B5B03001960) and (No.2018R1A5A7059549).

REFERENCES

[1] Lada A. Adamic and Eytan Adar. 2003. Friends and Neighbors on the Web. *Social Networks* 25, 3 (July 2003), 211–230.

[2] Paweena Chaiwanarom and Chidchanok Lursinsap. 2015. Collaborator Recommendation in Interdisciplinary Computer Science Using Degrees of Collaborative Forces, Temporal Evolution of Research Interest, and Comparative Seniority Status. *Knowledge-Based Systems* 75 (February 2015), 161–172.

[3] Hung-Hsuan Chen and C. Lee Giles. 2015. ASCOS++: An Asymmetric Similarity Measure for Weighted Networks to Address the Problem of SimRank. *ACM Transactions on Knowledge Discovery from Data, TKDD* 10, 2, Article 15 (October 2015), 26 pages.

[4] Quanyu Dai. 2019. *Implementation of DWNS.* Retrieved May 26, 2021 from https://github.com/wonniu/AdvT4NE_WWW2019

[5] Quanyu Dai, Xiao Shen, Liang Zhang, Qiang Li, and Dan Wang. 2019. Adversarial Training Methods for Network Embedding. In *Proceedings of the 28th International Conference on World Wide Web, WWW*. 329–339.

[6] Daniel Fogaras and Balazs Racz. 2005. Scaling Link-based Similarity Search. In *Proceedings of the 14th International Conference on World Wide Web, WWW*. 641–650.

[7] Aditya Grover. 2017. *Implementation of node2vec.* Retrieved May 26, 2021 from https://github.com/aditya-grover/node2vec

[8] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable Feature Learning for Networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM SIGKDD*. 855–864.

[9] Masoud Reyhani Hamedani and Sang-Wook Kim. 2017. JacSim: An Accurate and Efficient Link-Based Similarity Measure In Graphs. *Information Sciences* 414 (November 2017), 203–224.

[10] Masoud Reyhani Hamedani and Sang-Wook Kim. 2019. Pairwise Normalization in Simrank Variants: Problem, Solution, and Evaluation. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, ACM SAC*. 534–541.

[11] Jiawei Han, Micheline Kamber, and Jian Pei. 2006. *Data Mining: Concepts and Techniques, Second Edition.* Morgan Kaufmann, San Francisco.

[12] Roger A. Horn Han and Charles R. Johnson. 2013. *Matrix Analysis, Second Edition.* Cambridge University Press.

[13] Glen Jeh and Jennifer Widom. 2002. SimRank: A Measure of Structural-Context Similarity. In *Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, SIGKDD*. 538–543.

[14] Jinhong Jung, Namyong Park, Sael Lee, and U Kang. 2017. BePI: Fast and Memory-Efficient Method for Billion-Scale Random Walk with Restart. In *Proceedings of the 2017 ACM International Conference on Management of Data, SIGMOD*. 789–804.

[15] Jinhong Jung, Kijung Shin, Lee Sael, and U Kang. 2016. Random Walk with Restart on Large Graphs Using Block Elimination. *ACM Transactions on Database Systems, TODS* 41, 2, Article 12 (May 2016), 43 pages.

[16] Jérôme Kunegis, Julia Preusse, and Felix Schwager. 2013. What is the Added Value of Negative Links in Online Social Networks? In *Proceedings of the 22nd International Conference on World Wide Web, WWW*. 727–736.

[17] Jundong Li, Liang Wu, Ruocheng Guo, Chenghao Liu, and Huan Liu. 2019. Multi-Level Network Embedding with Boosted Low-Rank Matrix Approximation. In *Proceedings of the 2019 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining, ASONAM*. 49–56.

[18] Mo Li, Farhana M. Choudhury, Renata Borovica-Gajic, Zhiqiong Wang, Junchang Xin, and Jianxin Li. 2019. CrashSim: An Efficient Algorithm for Computing SimRank over Static and Temporal Graphs. In *Proceedings of the 36th IEEE International Conference on Data Engineering, IEEE ICDE*. 1141–1152.

[19] David Liben-Nowell and Jon Kleinberg. 2007. The Link-prediction Problem for Social Networks. *Journal of the American Society for Information Science and Technology, JASIST* 58, 7 (May 2007), 1–23.

[20] Zhenjiang Lin, Michael R. Lyu, and Irwin King. 2012. MatchSim: A Novel Similarity Measure Based on Maximum Neighborhood Matching. *Knowledge and Information Systems, KAIS* 32, 1 (July 2012), 141–166.

[21] Dmitry Lizorkin, Pavel Velikhov, Maxim Grinev, and Denis Turdakov. 2008. Accuracy Estimate and Optimization Techniques for SimRank Computation. In *Proceedings of the VLDB Endowment*. 422–433.

[22] Walid Magdy and Gareth J.Jones. 2010. PRES: A Score Metric for Evaluating Recall-oriented Information Retrieval Applications. In *Proceedings of the 33rd International Conference on Research and Development in Information Retrieval, ACM SIGIR*. 611–618.

[23] Christopher.D. Manning, Prabhakar Raghavan, and Hinrich Schutze. 2008. *Introduction to Information Retrieval.* Cambridge University Press.

[24] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. arXiv:1301.3781 [cs.CL]

[25] Bryan Perozzi. 2014. *Implementation of DeepWalk.* Retrieved May 26, 2021 from https://github.com/phanein/deepwalk

[26] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. DeepWalk: Online Learning of Social Representations. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, ACM SIGKDD*. 701–710.

[27] Tao Qin, Tie-Yan Liu, Jun Xu, and Hang Li. 2010. LETOR: A Benchmark Collection for Research on Learning to Rank for Information Retrieval. *Information Retrieval* 13, 4 (2010), 346–374.

[28] Jiezhong Qiu. 2018. *Implementation of NetMF.* Retrieved May 26, 2021 from https://github.com/xptree/NetMF

[29] Jiezhong Qiu, Yuxiao Dong, Hao Ma, Jian Li, Kuansan Wang, and Jie Tang. 2018. Network Embedding as Matrix Factorization: Unifying DeepWalk, LINE, PTE, and node2vec. In *Proceedings of the 11st ACM International Conference on Web Search and Data Mining, ACM WSDM*. 459–467.

[30] Y. Saad. 2003. *Iterative Methods for Sparse Linear Systems.* Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

[31] Henry Small. 1973. Co-citation in the Scientific Literature: A New Measure of the Relationship between Two Documents. *Journal of the American Society for Information Science and Technology, JASIST* 24, 4 (1973), 165–269.

[32] Jiankai Sun, Bortik Bandyopadhyay, Armin Bashizade, Jiongqian Liang, P. Sadayappan, and Srinivasan Parthasarathy. 2019. ATP: Directed Graph Embedding with Asymmetric Transitivity Preservation. In *Proceedings of the 33rd AAAI Conference on Artificial Intelligence, AAAI*. 265–272.

[33] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2014. Intriguing Properties of Neural Networks. arXiv:1312.6199 [cs.CV]

[34] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. LINE: Large-scale Information Network Embedding. In *Proceedings of the 24th International Conference on World Wide Web, WWW*. 1067–1077.

[35] Hanghang Tong, Christos Faloutsos, and Jia yu Pan. 2006. Fast Random Walk with Restart and Its Applications. In *Proceedings of the 6th IEEE International Conference on Data Mining, IEEE ICDM*. 613–622.

[36] Hongwei Wang, Jia Wang, Jialin Wang, Miao Zhao, Weinan Zhang, Fuzheng Zhang, Xing Xie, and Minyi Guo1. 2018. GraphGAN: Graph Representation Learning with Generative Adversarial Nets. In *Proceedings of the 32nd AAAI Conference on Artificial Intelligence, AAAI*. 2508–2515.

[37] Zhitao Wang, Chengyao Chen, and Wenjie Li. 2017. Predictive Network Representation Learning for Link Prediction. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM SIGIR*. 969–972.

[38] Jaewon Yang and Jure Leskovec. 2012. Defining and Evaluating Network Communities Based on Ground-Truth. In *Proceedings of the 12th IEEE International Conference on Data Mining, IEEE ICDM*. 745–754.

[39] Seok-Ho Yoon, Sang-Wook Kim, and Sunju Park. 2016. C-Rank: A Link-based Similarity Measure for Scientific Literature Databases. *Information Sciences* 326 (January 2016), 25–40.

[40] Weiren Yu, Xuemin Lin, Wenjie Zhang, Jian Pei, and Julie A. McCann. 2019. Simrank*: Effective and Scalable Pairwise Similarity Search Based on Graph Topology. *The VLDB Journal* 28, 3 (June 2019), 401–426.

[41] Weiren Yu, Wenjie Zhang, Xuemin Lin, Qing Zhang, and Jiajin Le. 2019. Accelerating Pairwise SimRank Estimation Over Static and Dynamic Graphs. *The VLDB Journal* 28, 1 (2019), 99–122.

[42] Peixiang Zhao, Jiawei Han, and Sun Yizhou. 2009. P-Rank: A Comprehensive Structural Similarity Measure over Information Networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management, ACM CIKM*. 553–562.