The VLDB Journal manuscript No.
(will be inserted by the editor)

Efficient Structural Node Similarity Computation on Billion-scale Graphs

Xiaoshuang Chen · Longbin Lai · Lu Qin · Xuemin Lin

Received: date / Accepted: date

Abstract Structural node similarity is widely used in analyzing complex networks. As one of the structural node similarity metrics, role similarity has the good merit of indicating automorphism (isomorphism). Existing algorithms to compute role similarity (e.g., RoleSim and NED) suffer from severe performance bottlenecks, and thus cannot handle large real-world graphs. In this paper, we propose a new framework, namely StructSim, to compute nodes' role similarity. Under this framework, we first prove that StructSim is an admissible role similarity metric based on the maximum matching. While the maximum matching is still too costly to scale, we then devise the BinCount matching that not only is efficient to compute but also guarantees the admissibility of StructSim. BinCount-based StructSim admits a precomputed index to query a single pair of node in $O(k \log D)$ time, where k is a small user-defined parameter and D is the maximum node degree. To build the index, we further devise an FM-sketch-based technique that can handle graphs with billions of edges. Extensive empirical studies show that StructSim performs much better than the existing works regarding both effectiveness and efficiency when applied to compute structural node similarities on the real-world graphs.

Keywords Node Similarity · Role Similarity · Efficiency · Link Analysis

Xiaoshuang Chen
University of New South Wales, Australia,
E-mail: xiaoshuang.chen@unsw.edu.au

Longbin Lai
Alibaba Group, China,
University of New South Wales, Australia,
E-mail: longbin.lailb@alibaba-inc.com

Lu Qin
Centre for AI, University of Technology Sydney, Australia,
E-mail: lu.qin@uts.edu.au

Xuemin Lin
University of New South Wales, Australia,
East China Normal University, China,
E-mail: lxue@cse.unsw.edu.au

1 Introduction

Structural node similarity is an important metric in graph analysis, and hence has attracted many attentions in the academia [18, 23, 24, 53, 60]. Among these efforts, role similarity [24] stands out because of the property of *automorphism confirmation*[1], that is the similarity between two nodes u and v is 1 (the maximum similarity value) if there is an automorphism from u to v. In this paper, given an *unlabelled* and *undirected* simple graph, we study the problem of computing nodes' structural similarity with automorphism confirmation, or more specifically, the role similarity. The term "role" literally refers to the *role* that a node plays in the graph. Examples of roles are a scholar's academic rank in the academia, enzymes in the PPI network [13], authorities in the web graph [26] and an individual's social position in the social network [36], to name a few. An effective way to infer the role of a node in a graph is to encompass the configuration of its neighbours. In this sense, we can evaluate the role similarity between two nodes by cross-comparing their contexts of neighborhood.

Applications. Role similarity is adopted in a wide scope of applications. In social science, role similarity can facilitate role discovery [51]. In bioinformatics, role similarity can be used to predict an unknown protein's

[1] It is isomorphism confirmation when computing similarity between nodes in two different graphs.

function, given that proteins with similar roles in the PPI network have similar functions [13]. In the world trade network, role similarity is useful for predicting the position of a country in the world system [44]. In a set of IP communication graphs of different time sequences, we can perform role analysis in one graph and leverage the role similarity to explore roles in the others [21]. Other applications include classifying or clustering nodes in the graphs [22], recommending nodes with similar roles [34], and detecting anomalous nodes [43].

State-of-the-art. Jin et al. [24] defined five properties for a role similarity metric, and a metric is called an *admissible role similarity metric* (or has *admissibility*) if it satisfies all the five properties. *Automorphism confirmation* is the most important one among these properties. The authors first proved that SimRank, a representative structural similarity measure, fails to guarantee the automorphism confirmation. Thus, SimRank is inappropriate to measure role similarity. The authors then proposed RoleSim as the first admissible role similarity metric. The RoleSim algorithm[2] inherits the framework of SimRank to compute the role similarities of *all pairs* of nodes in an iterative way.

Zhu et al. [60] proposed to compute structural node similarity by utilizing the node's *hierarchical structure* - the *k-adjacent tree*. The authors then proposed the NED distance metric between two nodes as the summation of the level-wise *tree edit distance* of the respective k-adjacent trees. Note that NED is originally a distance metric, and it can be trivially turned into a similarity metric via normalization. For consistency, we refer NED as the corresponding similarity metric in this paper, and we further prove that NED is an admissible role similarity metric as well (Section 3.2).

Motivations. RoleSim and NED are state-of-the-art algorithms to compute role similarity. However, both of them have severe performance bottlenecks, and hence only scale to million-scale graphs. RoleSim relies on a maximum matching of the neighbors to guarantee the automorphism confirmation, while it requires $O(kn^2d^2 \log d)$ time to compute the similarities of all pairs of nodes, where k is the number of iterations, n is the number of nodes, and d is the average degree. This apparently hinders applying RoleSim to large graphs. To speed up the computation, the authors further proposed IcebergRoleSim [25] that only computes node pairs whose similarity values are guaranteed to be larger than a given threshold. While this optimization, to some extent, boosts the algorithm, it does not improve the time complexity in the worst case. In addition to the efficiency issues, we observe the "odd-distance abnormality" of RoleSim (formally proved in Section 3.1), which shows that it behaves counter-intuitively in even-path graphs. Last but not least, RoleSim is constrained to the all-pairs computation due to the iterative framework, and thus one has to pay the all-pairs cost even when only interested in ad-hoc queries such as single-pair queries and single-source queries.

Unlike RoleSim, NED's hierarchical framework does not need the all-pairs cost for ad-hoc queries, but it is not scalable either. Specifically, NED pays $O(k\tau^3)$ time to query a single pair of nodes, where τ is the average number of nodes in each level of the k-adjacent trees. Note that NED's k-adjacent tree is based on the breadth-first search tree, but it involves the already-visited nodes on multiple levels. As a result, τ often increases exponentially regarding k. In the experiment, we witness $\tau > 100,000$ for certain nodes in a small graph with only 400 nodes when $k = 3$, and the cubic time complexity makes NED impossible to run.

Challenges. To resolve the scalability issues of existing algorithms, we need to tackle the following three main challenges.

Challenge I. We follow NED's hierarchical framework to free from RoleSim's all-pairs cost for ad-hoc queries, while the challenge is to seek a feasible hierarchical structure that can reflect the structural information and meanwhile is efficient to compute.

Challenge II. Challenge remains on how to compute the similarity based on the hierarchical structure such that the admissibility of role similarity metric [24] is satisfied.

Challenge III. Real-world graphs nowadays commonly reach billion scale, and it is challenging to compute role similarity at this scale.

Our Approaches. In this paper, we propose StructSim to address the above challenges as follows.

For Challenge I, we utilize a lightweight hierarchical structure called k-neighborhood subgraph $G_k(u)$ for each node u, in which nodes in the s^{th} level are nodes whose shortest distances to u are exactly s. Note that our k-neighborhood subgraph will only include each node once. Thus, it avoids the exponential growth of NED's k-adjacent trees.

For Challenge II, we devise the StructSim framework that computes the similarity of two nodes via the weighted average of level-wise similarities of the respective k-neighborhood subgraphs. In each level, the similarity value is measured based on a matching between nodes at that level. In this paper, we configure two matching methods for StructSim, namely maximum

[2] In the following, we will use the metric (e.g., RoleSim) and the algorithm to compute the metric interchangeably.

Table 1 Properties of RoleSim, IcebergRoleSim, NED and StructSim. Billion-scale indicates if each algorithm can handle billion-scale graphs.

Property	RoleSim	IcebergRoleSim	NED	StructSim
Single-pair Time Complexity	$O(kn^2d^2 \log d)$	$O(k\|H\|d^2 \log d)$	$O(k\tau^3)$	$O(k \log D)$
All-pairs Time Complexity	$O(kn^2d^2 \log d)$	$O(k\|H\|d^2 \log d)$	$O(kn^2\tau^3)$	$O(kn^2 \log D)$
Space Complexity	$O(n^2)$	$O(\|H\|)$	$O(\tau^2)$	$O(kn \log D)$
Billion-scale	×	×	×	✓

matching and BinCount matching. We formally prove that StructSim is an admissible role similarity metric under both configurations.

For Challenge III, we design two techniques for the StructSim framework, namely BinCount matching and FM-sketch-based index (FMS-Index). With the BinCount matching, the single-pair query can be conducted in $O(k \log D + nd)$ time, where the term nd indicates the time to construct the k-neighborhood subgraphs online for the query nodes. Note that the term nd can be trivially removed if we precompute the k-neighborhood subgraphs for all nodes as an index. Naively, this index can be built using the breadth-first search, which requires $O(n^2d)$ time, and is too costly to scale. Hence, we design the FMS-Index, which reduces the index building time to $O(rknd \log D)$, where r is the repeating times of performing FM-sketch (typically a small value). The construction of FMS-Index can be easily parallelized. Both the naive index and FMS-Index occupy $O(kn \log D)$ space, while it allows the all-pairs similarity computation without paying $O(n^2)$ space as RoleSim to maintain the similarity values. These techniques eventually empower StructSim to handle graphs of billions of edges.

Apart from the aforementioned efficiency gains, we further conduct extensive case studies, which demonstrate that StructSim yields more reasonable results than both RoleSim (and its extension IcebergRoleSim) and NED. Table 1 summarizes the properties of RoleSim, IcebergRoleSim, NED and our StructSim. Specifically, StructSim is the only one that can handle billion-scale graphs.

Contributions. Our contributions are summarized as follows.

(1) A new hierarchical framework named StructSim *to compute role similarity.* We propose StructSim that follows the hierarchical framework to better support ad-hoc queries.

(2) Flexible configurations of StructSim *with the guarantee of admissibility.* The StructSim framework is designed to allow two different matching methods (i.e., the maximum matching and the BinCount matching), which are both proved to guarantee the admissibility of role similarity metric.

(3) Efficient matching algorithm and indexing technique that scale to graphs with billions of edges. We devise the BinCount matching and FM-sketch-based index (FMS-Index) for StructSim, which empower StructSim to handle billion-scale graphs. To the best of our knowledge, StructSim is the first algorithm that can compute nodes' role similarity on graphs at billion scale.

(4) Extensive experimental studies. We conduct extensive experiments that show StructSim is significantly faster than the existing algorithms. For example, on the Astroph dataset, to answer a single-pair similarity query, StructSim spends less than 1 ms without the index (less than 10 μs with the index), while NED needs more than 150 seconds. For the all-pairs similarity computation, both NED and RoleSim cannot terminate within one day. IcebergRoleSim needs nearly three hours to finish the computation, while the proposed StructSim only spends less than 100 seconds. We also conduct comprehensive case studies to show that StructSim achieves better result qualities.

Organization. The rest of this paper is organized as follows. Section 2 gives preliminaries and formally defines the research problem; Section 3 reviews the existing works of RoleSim, IcebergRoleSim and NED. Besides, we theoretically prove the "odd-distance abnormality" of RoleSim and the admissibility of NED, which extends our conference paper [11]; Section 4 introduces the StructSim framework and proves its admissibility with the maximum matching; Section 5 describes two techniques, namely the BinCount matching and FM-sketch-based index, to compute StructSim efficiently. We prove the admissibility of StructSim with the BinCount matching as an extension of our conference paper, and the FM-sketch-based indexing technique and its parallelization are first proposed in this journal paper; Section 6 reports case studies and the experimental results on real-world graphs, which involves more results and analysis than our conference paper; Section 7 summarizes the related work and Section 8 concludes this paper.

Table 2 Table of Notations

Notation	**Description**
G	the input graph
$G_k(u)$	the k-neighborhood subgraph of node u
n	the number of nodes in G
m	the number of edges in G
d_u	the degree of node u
d	the average node degree in G
D	the maximum node degree in G
$N(u)$	the neighbors of node u
$N_i(u)$	the i-hop neighbors of node u
$N_{\leq i}(u)$	the i-hop reachable neighbors of node u
$M_i(u,v)$	the matching between $N_i(u)$ and $N_i(v)$

2 PRELIMINARIES

In this paper, we consider the undirected and unlabelled simple graph $G=(V,E)$, where V is the node set and E is the edge set. We denote the number of nodes $|V|$ and the number of edges $|E|$ by n and m respectively. For a node $u \in V$, $N(u)$ denotes the neighbors of u, i.e., $N(u)=\{v \in V|(u,v)\in E\}$, and d_u denotes the degree of node u, i.e., $d_u=|N(u)|$. Specifically, we use d and D to denote the average node degree and the maximum node degree respectively. Given two nodes u and v, we denote $\delta(u,v)$ as the *distance* between u and v, namely the length of the shortest path from u to v. Given a subset of the nodes $U \subseteq V$, the *induced graph* $G(U)$ is defined as a subgraph of G formed by the nodes in U and all edges among U, namely $G(U)=(U,E(U))$, where $E(U)=\{(u,v)|u\in U, v\in U \wedge (u,v)\in E\}$. Table 2 summarizes the notations in this paper, and we further have the following definitions.

Definition 1 (i-HOP NEIGHBORS) The i-hop neighbors of node u, denoted as $N_i(u)$, contains all the nodes whose distance to u are of length i ($i \geq 0$), namely, $N_i(u)=\{v\in V|\delta(u,v)=i\}$. Specifically, $N_0(u)=\{u\}$ and $N_1(u)=N(u)$.

Definition 2 (i-HOP REACHABLE NEIGHBORS) The i-hop reachable neighbors of node u, denoted as $N_{\leq i}(u)$, contains all the nodes who have a path to u with length no more than i ($i \geq 0$). Clearly, $N_{\leq i}(u)=\bigcup_{j=0}^{i} N_j(u)$.

Definition 3 (K-NEIGHBORHOOD SUBGRAPH) The k-neighborhood subgraph of node u is the induced subgraph $G(N_{\leq k}(u))$ (or $G_k(u)$ for short).

Definition 4 (MATCHING OF i-HOP NEIGHBORS) We define the **matching** between two node sets S_1 and S_2, denoted as $M(S_1,S_2)$, as $M(S_1,S_2)=\{(x,y)|x\in S_m \wedge y=f(x)\in S_M\}$. Here, $f: S_m \rightarrow S_M$ is an injective function, where S_m is the smaller set and S_M is the larger set of S_1, S_2. Given node u and node v, we further denote $M_i(u,v)$ as the matching of the respective i-hop neighbors $N_i(u)$ and $N_i(v)$, namely $M_i(u,v)=M(N_i(u),N_i(v))$. Specifically, when $i=0$, $M_0(u,v)=\{(u,v)\}$. In this paper, we denote $M(u,v)=M_1(u,v)$ as the matching of the neighbors.

Definition 5 (GRAPH ISOMORPHISM AND AUTOMORPHISM) An isomorphism of graph $G=(V_G,E_G)$ and graph $H=(V_H,E_H)$ is a bijective mapping between V_G and V_H, denoted by $\sigma: V_G \rightarrow V_H$, such that for any two nodes u and v in G, $(u,v)\in E_G$ if and only if $(\sigma(u),\sigma(v))\in E_H$. Specially, automorphism is an isomorphism mapping G to itself.

Definition 6 (AUTOMORPHIC EQUIVALENCE) For node u and node v in a graph G, if there is an automorphism σ of G satisfying $\sigma(u)=v$, we say that u and v are automorphically equivalent, denoted as $u\equiv v$.

As a node similarity metric that is inferred from the structural context, *role similarity metric* [24] has the following five properties.

Definition 7 (ROLE SIMILARITY METRIC PROPERTIES) For a graph $G=(V,E)$, a similarity metric $\mathsf{Sim}(u,v)$ that measures the role similarity between node u and node v should satisfy:

P1. Range: $\forall u,v\in V, 0\leq \mathsf{Sim}(u,v)\leq 1$.
P2. Symmetry: $\forall u,v\in V, \mathsf{Sim}(u,v)=\mathsf{Sim}(v,u)$.
P3. Automorphism confirmation: *If* $u\equiv v, \mathsf{Sim}(u,v)=1$, where $u\equiv v$ denotes that u and v are automorphically equivalent.
P4. Transitive similarity: $\forall w\in V$, $\mathsf{Sim}(u,w)=\mathsf{Sim}(v,w)$ if $u\equiv v$.
P5. Triangle inequality: $\forall u,v,w\in V$, $\mathsf{Dist}(u,v)\leq \mathsf{Dist}(u,w)+\mathsf{Dist}(v,w)$, where $\mathsf{Dist}(u,v)=1-\mathsf{Sim}(u,v)$.

According to [24], a similarity measure is an **admissible role similarity metric** if it satisfies all the five properties.

3 Existing Works

RoleSim and NED are the two state-of-the-art algorithms to compute role similarity scores that satisfy all the five properties in Definition 7. In this section, we first overview RoleSim [24] and its variant IcebergRoleSim [25] in Section 3.1, and give a formal proof for the observation of "odd-distance abnormality". Then, we introduce NED [60], and prove its admissibility of being a role similarity metric in Section 3.2.

3.1 RoleSim

RoleSim is based on the iterative computation framework as SimRank. Given a graph $G = (V, E)$, the RoleSim similarity of node pair $(u, v) \in V \times V$ is computed as:

$$\mathsf{RoleSim}^k(u,v) = \beta \max_{M(u,v)} \frac{\sum_{(x,y)\in M(u,v)} \mathsf{RoleSim}^{k-1}(x,y)}{max(|N(u)|,|N(v)|)} + (1-\beta), \tag{1}$$

where $\mathsf{RoleSim}^k(u,v)$ denotes the $\mathsf{RoleSim}(u,v)$ value in the k^{th} iteration, $M(u,v)$ is the matching between $N(u)$ and $N(v)$, and $0 < \beta < 1$ is a decay factor. According to Equation 1, $\mathsf{RoleSim}^k(u,v)$ is updated as the *maximum matching* between $N(u)$ and $N(v)$ regarding the respective RoleSim values in the previous iteration. While the maximum matching is indispensable for RoleSim to meet with the automorphism confirmation property (Definition 7), it is in general a costly operation. Specifically, the time complexity of RoleSim is $O(kn^2d^2\log d)$ when using the most efficient greedy algorithm [5] to compute the maximum matching. It is apparently too costly to apply to large real-world graphs.

Jin et al. then proposed IcebergRoleSim [25] to speed up the computation of RoleSim. To do so, they maintain the node pairs whose RoleSim values are guaranteed to be larger than a given threshold θ in a table H. After that, IcebergRoleSim will only update the similarity values for the pairs in H. As for those pruned, an initial value is given before the computation. To be specific, the time complexity of IcebergRoleSim is $O(k|H|d^2\log d)$, with $|H| \le n^2$. However, IcebergRoleSim only optimizes RoleSim heuristically, and it cannot improve the complexity in the worst case and thus has limited performance. Moreover, both RoleSim and IcebergRoleSim are constrained to the all-pairs similarity computation due to the iterative framework, and one has to pay the all-pairs cost even when ad-hoc queries are of interest.

Odd-distance Abnormality. In addition to the above drawbacks, we observe the following "odd-distance abnormality" on even paths for RoleSim (and IcebergRoleSim). In Figure 1, we show an even path P of 99 nodes, in which u_{50} is the middle node. In terms of nodes' role similarity, one would expect that the closer nodes in the path should have higher role similarity scores, e.g. $\mathsf{Sim}(u_1,u_2) > \mathsf{Sim}(u_1,u_4) > \mathsf{Sim}(u_1,u_6) > \cdots > \mathsf{Sim}(u_1,u_{50})$ in Figure 1. However, RoleSim always gives the same values for these

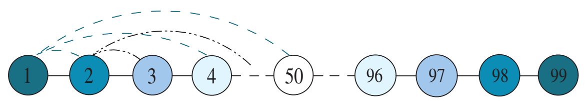


Fig. 1 Even path P, u_{50} is the middle node. Nodes of the same filling color are automorphically equivalent.

node pairs, e.g. $\mathsf{RoleSim}(u_1,u_2) = \mathsf{RoleSim}(u_1,u_4) = \cdots = \mathsf{RoleSim}(u_1,u_{50})$. Besides, RoleSim also renders $\mathsf{RoleSim}(u_2,u_3) = \mathsf{RoleSim}(u_2,u_5) = \cdots = \mathsf{RoleSim}(u_2,u_{49})$, and so forth. As the distances of the abnormal node pairs are odd, we call this phenomenon "odd-distance abnormality". Next, we define the "odd-distance abnormality" problem and then give a formal proof.

Theorem 1 (Odd-Distance Abnormality) *Let P be an even path $u_1 - u_2 - \cdots - u_t - \cdots - u_{n-1} - u_n$, where $n = 2t-1$ and $t \ge 4$. For every node u_i, $i \in [1, t-3]$ on P, it holds that $\forall d \in [1, \frac{t-i-1}{2}]$, $\mathsf{RoleSim}(u_i,u_{i+1}) = \mathsf{RoleSim}(u_i,u_{i+2d+1})$.*

Proof. We assume $\beta = 0$ for the RoleSim computation [24]. Actually, the value of β has no impact on the proof. We denote $(i,j)^k$ as the RoleSim value of the node pair, (u_i,u_j) at iteration k. We prove Theorem 1 by performing mathematical induction on two conditions listed as follows. Specifically, condition 2 indicates an order sequence, that for $\forall k$, $(t,t-1)^k \ge (t-1,t-2)^k \ge (t-2,t-3)^k \ge \cdots \ge (2,1)^k$ holds.

Condition 1: $\forall k, 1 \le i \le t-3$, $1 \le d \le \frac{t-i-1}{2}$, $(i,i+1)^k = (i,i+2d+1)^k$;
Condition 2: $\forall k, 3 \le i \le t, (i,i-1)^k \ge (i-1,i-2)^k$.

Induction Basis: Based on the definition and initialization of RoleSim in [24], it is easy to prove that the condition 1 and condition 2 hold at 0^{th} iteration.

Inductive Step: Assume the above two conditions hold at $(k-1)^{th}$ iteration, we further prove that they are also true at k^{th} iteration.

Firstly, we prove condition 1 is true at the k^{th} iteration. The proof for this condition includes two cases:

Case 1: $i = 1$. Based on the assumption, $(2,3)^{k-1} \ge (2,1)^{k-1}$, thus we have $(1,2)^k = \frac{(2,3)^{k-1}}{2}$. Similarly, we have that for $\forall d \in [1, \frac{t-2}{2}]$,

$$(1,2d+2)^k = max\{\frac{(2,2d+1)^{k-1}}{2}, \frac{(2,2d+3)^{k-1}}{2}\}$$

If $2d+3 \le t$ is true, then we can get $(1,2d+2)^k = \frac{(2,3)^{k-1}}{2}$ as $(2,3)^{k-1} = (2,2d+1)^{k-1} = (2,2d+3)^{k-1}$. Otherwise, if $2d+3 = t+1$, then we have $(2,2d+3)^{k-1} = (2,2d+1)^{k-1}$ due to the automorphic equivalence. Therefore, $(1,2d+2)^k = \frac{(2,3)^{k-1}}{2} = (1,2)^k$ holds. And for node u_1, the condition 1 holds.

Case 2: $1 < i \le t-3$. Based on Equation 1, for $\forall d \in [1, \frac{t-i-1}{2}]$, we have the following two equations.

$$(i,i+1)^k = max\{\frac{(i-1,i)^{k-1} + (i+1,i+2)^{k-1}}{2}, \frac{(i-1,i+2)^{k-1} + (i+1,i)^{k-1}}{2}\}$$

$$(i,i+2d+1)^k = max\{ \frac{(i-1,i+2d)^{k-1} + (i+1,i+2d+2)^{k-1}}{2}, \frac{(i-1,i+2d+2)^{k-1} + (i+1,i+2d)^{k-1}}{2}\}$$

According to $(i-1,i)^{k-1} = (i-1,i+2)^{k-1}$ and $(i+1,i+2)^{k-1} \ge (i+1,i)^{k-1}$, we have $(i,i+1)^k = \frac{(i-1,i)^{k-1}+(i+1,i+2)^{k-1}}{2}$. Similarly, we can get $(i,i+2d+1)^k = \frac{(i-1,i)^{k-1}+(i+1,i+2)^{k-1}}{2}$ based on the two conditions for $(k-1)^{th}$ iteration. The corner case of this part for $i+2d+2 = t+1$ is similar to that in Case 1. Therefore, case 2 holds. Combining Case 1 and Case 2, the first condition holds at the k^{th} iteration.

Secondly, we prove that the second condition holds at the k^{th} iteration. The proof contains three cases as follows.

Case 1: $i = t$. We have

$$(t,t-1)^k = max\{\frac{(t+1,t)^{k-1} + (t-1,t-2)^{k-1}}{2}, \frac{(t+1,t-2)^{k-1} + (t-1,t)^{k-1}}{2}\}$$

$$(t-1,t-2)^k = max\{\frac{(t,t-1)^{k-1} + (t-2,t-3)^{k-1}}{2}, \frac{(t,t-3)^{k-1} + (t-2,t-1)^{k-1}}{2}\}$$

As node u_{t-1} and node u_{t+1} are automorphically equivalent, then we can get $(t,t-1)^k = \frac{(t-1,t)^{k-1}+(t-1,t-2)^{k-1}}{2}$. Based on the assumption, $(t-2,t-3)^{k-1} = (t,t-3)^{k-1}$ and $(t,t-1)^{k-1} \ge (t-2,t-1)^{k-1}$ hold, so we immediately have $(t-1,t-2)^k = \frac{(t,t-1)^{k-1}+(t-2,t-3)^{k-1}}{2}$. Finally, we can get the conclusion that $(t,t-1)^k \ge (t-1,t-2)^k$ holds as $(t-1,t-2)^{k-1} \ge (t-2,t-3)^{k-1}$ is true.

Case 2: $3 < i < t$. In this scenario, we have

$$(i,i-1)^k = max\{\frac{(i+1,i)^{k-1} + (i-1,i-2)^{k-1}}{2}, \frac{(i+1,i-2)^{k-1} + (i-1,i)^{k-1}}{2}\}$$

$$(i-1,i-2)^k = max\{\frac{(i,i-1)^{k-1} + (i-2,i-3)^{k-1}}{2}, \frac{(i,i-3)^{k-1} + (i-1,i-2)^{k-1}}{2}\}$$

Since $(i-1,i-2)^{k-1} = (i+1,i-2)^{k-1}$ and $(i+1,i)^{k-1} \ge (i-1,i)^{k-1}$, we have $(i,i-1)^k = \frac{(i+1,i)^{k-1}+(i-1,i-2)^{k-1}}{2}$. Similarly, we can derive $(i-1,i-2)^k = \frac{(i,i-1)^{k-1}+(i-2,i-3)^{k-1}}{2}$. From the assumption, $(i+1,i)^{k-1} \ge (i,i-1)^{k-1}$ and $(i-1,i-2)^{k-1} \ge (i-2,i-3)^{k-1}$, therefore, $\forall i \in (3,t), (i,i-1)^k \ge (i-1,i-2)^k$ holds, which can directly derive $(t-1,t-2)^k \ge (t-2,t-3)^k \ge \cdots \ge (4,3) \ge (3,2)$

Case 3: $i = 3$. We have $(3,2)^k = \frac{(2,1)^{k-1}+(3,4)^{k-1}}{2}$ and $(2,1)^k = \frac{(2,3)^{k-1}}{2}$. Since $(4,3)^{k-1} \ge (2,3)^{k-1}$, $(3,2)^k \ge (2,1)^k$ holds.

Combining Case 1, Case 2, and Case 3, the second condition also holds at the k^{th} iteration.

Hence, condition 1 and condition 2 hold at every iteration. Condition 1 indicates the "Odd-Distance Abnormality" issue. □

Remark 1 In Section 6.2, we give a case study on an even-path graph to demonstrate the "odd-distance abnormality" of RoleSim. While it is hard to generalize the proof of "odd-distance abnormality" to an arbitrary graph, we speculate that such abnormality may produce negative effects on a wider basis. In Section 6.2, we conduct some other case studies, in which there is one conducted on the Barbell graph that is a regular extension of the path graphs. According to the case study, RoleSim produces counter-intuitive results on the Barbell graph, and it is also less effective than both NED and our proposed StructSim on real-life graphs.

3.2 NED

Different from RoleSim based on the iterative computation framework, NED provides another perspective to compute role similarity [60]. The intuition of NED is that two nodes are similar if their respective hierarchical structures are similar. To describe such hierarchical structure, NED adopts the k-adjacent tree as a "signature" to represent each node. A k-adjacent tree rooted at node u is the k-level breadth-first search tree from node u (with the already-visited nodes present). Given the k-adjacent trees of two nodes u and v, NED(u,v) is evaluated as the tree edit distance of the respective trees. Note that NED is originally a distance metric. For consistency, we refer NED in this paper the corresponding similarity metric based on a trivial normalization as: NED $= 1 - \frac{\text{NED}^*}{\text{NED}^*_{max}}$, in which NED* is the

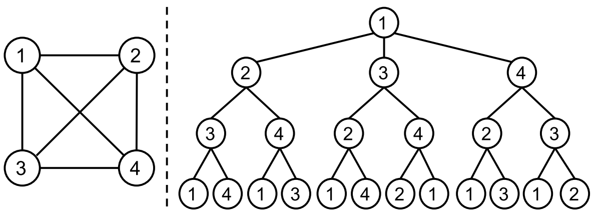


Fig. 2 A four complete graph and the NED's 3-adjacent tree for node u_1

original distance metric and NED^*_{max} is the maximum NED^*. Next, we prove that NED is also an admissible role similarity metric that satisfies all the five properties in Definition 7.

Theorem 2 NED *is an admissible role similarity metric.*

Proof. Refer to Definition 7, property *P1* can be trivially proved. Property *P2* and property *P5* can be easily verified as NED^* is a distance metric [60].

Property *P3* holds, as if nodes u and v are automorphically equivalent ($u \equiv v$), their respective k-adjacent trees must be isomorphic, leading to $\mathsf{NED}^*(u, v) = 0$ and $\mathsf{NED}(u, v) = 1$.

For property *P4*, we have $\mathsf{NED}^*(u, w) + \mathsf{NED}^*(u, v) \geq \mathsf{NED}^*(v, w)$ and $\mathsf{NED}^*(v, w) + \mathsf{NED}^*(u, v) \geq \mathsf{NED}^*(u, w)$ based on property *P5*; according to *P3*, it satisfies that if $u \equiv v$, then $\mathsf{NED}^*(u, v) = 0$. Therefore, we can get that $\mathsf{NED}^*(u, w) \geq \mathsf{NED}^*(v, w)$ and $\mathsf{NED}^*(v, w) \geq \mathsf{NED}^*(u, w)$, i.e. $\mathsf{NED}^*(u, w) = \mathsf{NED}^*(v, w)$. Thus, *P4* also holds. □

Alternatively, we can use NED to measure the role similarities. However, the bottleneck of NED remains to compute the tree edit distance of the k-adjacent trees, which is in general NP-Complete [57]. In order to tackle this bottleneck, the authors then proposed TED* [60], a relaxation of tree edit distance that is polynomially solvable, leading to $O(k\tau^3)$ time for answering a single-pair query, in which τ is the average number of nodes in one tree level. Note that τ grows exponentially with k and can be huge in practice. We present in Figure 2 one of the 3-adjacent trees of a four complete graph, which clearly shows the exponentially growing trend of the tree. The problem attributes to the involvement of the already-visited nodes in the breadth-first search in the k-adjacent tree. Even when we use a small k value as suggested by the authors, τ, and hence the query time, can still be unbounded in real-world graphs. In the experiment, we witness a small graph of 400 nodes in which some nodes have generated the 3-adjacent trees with more than $100,000$ nodes at a certain level, and the computation fails naturally due to the cubic time complexity.

4 The StructSim Framework

As both RoleSim and NED are inefficient to compute, we propose StructSim in this paper to compute role similarity based on the hierarchical k-neighborhood subgraphs. In the following, we first introduce the definition of StructSim, and present the StructSim computation algorithm by showing how k-neighborhood subgraph is leveraged to compute similarity. Then, we discuss the first configuration that makes StructSim an admissible role similarity metric. Finally, we analyze the complexity of the algorithm and propose several optimization perspectives.

4.1 StructSim Overview

We define StructSim that computes the role similarity score of a node pair via the weighted average of level-wise similarities between the respective k-neighborhood subgraphs. Specifically,

Definition 8 (StructSim) Given an undirected and unlabelled simple graph $G = (V, E)$, the StructSim between two nodes $u \in V$ and $v \in V$, denoted as $\mathsf{StructSim}(u, v)$, is calculated as

$$\mathsf{StructSim}(u, v) = \tilde{\omega_0}\zeta(u, v) + \sum_{i=1}^{k} \frac{\tilde{\omega_i} \sum_{(x,y)\in M_i(u,v)} f(x, y)}{max(|N_i(u)|, |N_i(v)|)},$$

where $\zeta(u, v) = \frac{min(d_u,d_v)}{max(d_u,d_v)}$ gives an initial score of u and v, $M_i(u, v)$ is a matching (Definition 4) between $N_i(u)$ and $N_i(v)$, $f : (x, y) \to [0, 1]$ is a predefined function to compute similarity values of a node pair in the matching, and $\tilde{\omega_i}$ is a hop-dependent weighting parameter satisfying $\sum_{i=0}^{k} \tilde{\omega_i} = 1$ (see the following algorithm for more details).

Remark 2 Here we discuss the options of initialization (i.e., $\zeta(\cdot)$) for StructSim computation. Refer to RoleSim [24], there are two kinds of initialization functions, i.e., degree-binary initialization ($\zeta(u, v) = 1$ if $d_u = d_v$ otherwise $\zeta(u, v) = 0$) and degree-ratio initialization ($\zeta(u, v) = \frac{min(d_u,d_v)}{max(d_u,d_v)})^3$, which can be used to compute role similarity scores. Our StructSim can adopt both of the two initialization functions as well. In this paper, we use the degree-ratio by default since it gives

[3] In [24], RoleSim has a third initialization, namely "ALL-1" initialization, which renders same similarity scores as the degree-ratio initialization.

initial scores in a more fine-grained manner than the degree-binary initialization. We will compare the performance of StructSim with different initialization functions in Section 6.2.

Algorithm. We outline the algorithm to compute StructSim in Algorithm 1. First of all, we assign the initial similarity value of the node pair in line 1. Then, we compute StructSim score based on the respective k-neighborhood subgraphs (line 2-line 12). Specifically, we deploy two operations - `Matching` and `Delta` - to dynamically update the similarity value level by level. At each level i, we can get the i-hop neighbours $N_i(u)$ and $N_i(v)$ from their k-neighborhood subgraphs (line 3). The `Matching` operation (line 10), as the name suggests, finds a matching $M_i(u,v)$ of $N_i(u)$ and $N_i(v)$. Given $M_i(u,v)$, the `Delta` operation (line 11) computes the delta value of current level $\Delta_i(u,v)$, which is a summation of $f(x,y)$ for each $(x,y) \in M_i(u,v)$. Finally, the similarity value is updated as a weighted average between the old value and $\Delta_i(u,v)$ (line 12). The weight $0 < \omega_i < 1$ is a parameter used to adjust the contribution of each level, and the weighting parameter $\tilde{\omega_i}$ in Definition 8 is accordingly computed by $\tilde{\omega_i} = \prod_{j=i+1}^{k}(1-\omega_j)\omega_i$. Specifically, we set $\omega_i = \frac{df}{1+i}$ and if $df < 1$, the value of $\tilde{\omega_i}$ gradually decreases regarding i, which conforms with the intuition that nodes closer to the target node should contribute more to the final similarity score. Note that $N_i(u)$ or $N_i(v)$ may be empty at a certain level. In this case, if both are empty, we stop the iteration and return the similarity value (line 5); otherwise, we simply let $\Delta_i(u,v) = 0$ (line 8) and continue the loop until k, which naturally penalizes the final similarity in line 12.

Remark 3 StructSim follows NED's hierarchical framework rather than RoleSim's iterative scheme to better support ad-hoc queries. Surprisingly, we observe that the hierarchical framework often renders better ranking quality than the iterative one. In Theorem 1, we have shown that RoleSim behaves anti-intuitively in even-path graphs. In Section 6.2, we further perform different case studies, which show that the hierarchical framework yields better ranking qualities. In addition, it is also possible to make StructSim iterative by setting $f(x,y)$ in line 11 of Algorithm 1 to the StructSim value of (x,y) in the previous iteration. However, such iterative scheme will use the information of k-hop neighbors multiple times, which will make it perform poorly in real-world tasks.

The `Matching` and `Delta` are critical operations for StructSim, which can affect both the admissibility and performance of the algorithm.

Algorithm 1: Framework for StructSim Computation

```
Input  : The graph $G$, a node pair $(u,v)$ and parameter $k$.
Output : StructSim$(u,v)$.
1  $S \leftarrow \frac{min(d_u,d_v)}{max(d_u,d_v)}$;
2  foreach level $i \in [1,k]$ do
3      Get $i$-hop neighbours $N_i(u)$ and $N_i(v)$ from $G_k(u)$ and $G_k(v)$;
4      if $N_i(u) = \emptyset$ and $N_i(v) = \emptyset$ then
5          break;
6      else
7          if $N_i(u) = \emptyset$ or $N_i(v) = \emptyset$ then
8              $\Delta_i(u,v) \leftarrow 0$;
9          else
10             (Matching) Computing the matching $M_i(u,v)$;
11             (Delta) $\Delta_i(u,v) \leftarrow \frac{\sum_{(x,y)\in M_i(u,v)} f(x,y)}{max(|N_i(u)|,|N_i(v)|)}$;
12         $S \leftarrow (1-\omega_i)S + \omega_i \Delta_i(u,v)$;
13 return $S$.
```

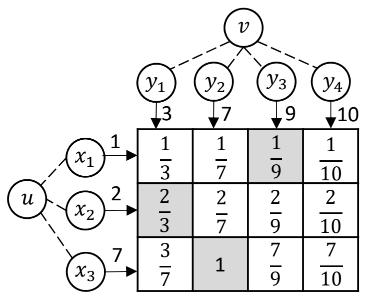
4.2 Maximum Matching

Our primary goal is to guarantee the admissibility of StructSim according to Definition 7. Inspired by RoleSim, our first attempt is **maximum matching**, and we accordingly call the algorithm StructSim-Max. Specifically, we compute the matching in line 10 that maximizes the delta value in line 11. Here we use the initial scores as the predefined similarity value, namely $f(x,y) = \zeta(x,y) = \frac{min(d_x,d_y)}{max(d_x,d_y)}$. Intuitively, we want to match the nodes that are most similar regarding their degrees. We denote the maximum matching at the i^{th} level of u,v as $M_i^{max}(u,v)$. Below we give an example to show how the maximum matching is processed.

Example 1 Given a node pair (u,v) $(u,v \in V)$, and their i-hop neighbours $N_i(u) = \{x_1,x_2,x_3\}$ and $N_i(v) = \{y_1,y_2,y_3,y_4\}$, let their degrees be $d_{x_1} = 1, d_{x_2} = 2, d_{x_3} = 7$, and $d_{y_1} = 3, d_{y_2} = 7, d_{y_3} = 9$, $d_{y_4} = 10$. Figure 3 plots the corresponding maximum matching process (the number near the arrow is the node's degree). For example, nodes x_2, x_3 match with y_1, y_2, respectively. As for node x_1, it matches with y_3 as y_1 and y_2 has been matched to others and cannot be reused. Consequently, $M_i^{max}(u,v) = \{(x_1,y_3),(x_2,y_1),(x_3,y_2)\}$ as shown in the shadowed grids, and $\Delta_i(u,v) = \frac{\frac{2}{3}+\frac{7}{7}+\frac{1}{9}}{4} = 0.45$.

With the maximum matching, we can write StructSim using the following equation:

$$\text{StructSim}(u,v) = \sum_{i=0}^{k} \frac{\tilde{\omega_i} \sum_{(x,y)\in M_i^{max}(u,v)} f(x,y)}{max(|N_i(u)|,|N_i(v)|)}, \quad (2)$$



v
y_1 y_2 y_3 y_4
3 7 9 10
u
x_1 x_2 x_3
1 2 7
$\frac{1}{3}$ $\frac{1}{7}$ $\frac{1}{9}$ $\frac{1}{10}$
$\frac{2}{3}$ $\frac{2}{7}$ $\frac{2}{9}$ $\frac{2}{10}$
$\frac{3}{7}$ 1 $\frac{7}{9}$ $\frac{7}{10}$

Fig. 3 The maximum matching process of $N_i(u)$ and $N_i(v)$

where $f(x,y) = \frac{min(d_x,d_y)}{max(d_x,d_y)}$ and $\tilde{\omega_i} = \prod_{j=i+1}^{k}(1-\omega_j)\omega_i$. Specifically, $\omega_i = \frac{df}{i+1}$, $\sum_{i=0}^{k}\tilde{\omega_i} = 1$ and $0 < df \leq 1$ is a damping factor. We next prove the admissibility of StructSim-Max by proving that the StructSim in Equation 2 satisfies all the properties in Definition 7.

Theorem 3 (Admissibility) StructSim *with the maximum matching is an admissible role similarity metric.*

Proof. We prove this theorem by showing that if $f(x,y)$ is an admissible role similarity metric, then StructSim in Equation 2 is an admissible role similarity metric as well. The admissibility of $f(x,y) = \frac{min(d_x,d_y)}{max(d_x,d_y)}$ has been proved in [24]. Next, we prove the admissibility of StructSim. Given $\Delta_i(u,v) = \frac{\sum_{(x,y)\in M_i^{max}(u,v)} f(x,y)}{max(|N_i(u)|,|N_i(v)|)}$ in Equation 2, we need to prove that for $\forall i \in [1,k]$, Δ_i satisfies all the five properties in Definition 7. It is not hard to verify that Δ_i satisfies the properties of Range, Symmetry, and Automorphic confirmation (Definition 7). We next prove that Δ_i satisfies the other two properties, i.e., Transitive similarity and Triangle inequality.

We first prove if $u \equiv v$, $\Delta_i(u,w) = \Delta_i(v,w)$. When $u \equiv v$, there exists a one-to-one equivalence, denoted as σ, between nodes in $N_i(u)$ and $N_i(v)$. Accordingly, for $M_i^{max}(u,w) = \{(x,y)|x \in N_i(u), y \in N_i(w)\}$, we define the matching $M_i(v,w) = \{(\sigma(x),y)|\forall(x,y) \in M_i^{max}(u,w)\}$. It is not hard to verify that $M_i(v,w)$ is a maximum matching between $N_i(v)$ and $N_i(w)$. As $f(x,y)$ is admissible, we have $f(x,y) = f(\sigma(x),y)$ and thus $\Delta_i(u,w) = \Delta_i(v,w)$.

Then, we prove $1-\Delta_i(u,v) \leq 1-\Delta_i(u,w)+1-\Delta_i(v,w)$. The proof includes three cases: (1) $|N_i(w)| \leq |N_i(v)| \leq |N_i(u)|$; (2) $|N_i(v)| \leq |N_i(w)| \leq |N_i(u)|$; and (3) $|N_i(v)| \leq |N_i(u)| \leq |N_i(w)|$. We give the proof of case 1 while the proofs of the other cases are similar. We define a matching between $N_i(u)$ and $N_i(v)$ as $M_i(u,v) = \{(x,z)|(x,y) \in M_i^{max}(u,w) \bigwedge (z,y) \in M_i^{max}(v,w)\}$, and $W(\cdot)$ as a function computed by $W(S) = \sum_{(x,y)\in S} f(x,y)$ in which S denotes a set of node pairs. Note that $|M_i^{max}(u,w)| = |M_i^{max}(v,w)| = |N_i(w)|$ and $W(M_i^{max}(u,v)) \geq W(M_i(u,v))$, we have

$$
\begin{aligned}
&1-\Delta_i^I(u,w)+1-\Delta_i^I(v,w)-(1-\Delta_i^I(u,v))\\
&= -\Delta_i^I(u,w)-\Delta_i^I(v,w)+\Delta_i^I(u,v)+1\\
&\geq -\frac{W(M_i^{max}(u,w))}{|N_i(u)|} - \frac{W(M_i^{max}(v,w))}{|N_i(v)|} + \frac{W(M_i(u,v))}{|N_i(u)|} + 1\\
&\geq \frac{|N_i(w)| - W(M_i^{max}(u,w))}{|N_i(u)|} + \frac{|N_i(w)| - W(M_i^{max}(v,w))}{|N_i(v)|}\\
&- \frac{|N_i(w)| - W(M_i(u,v))}{|N_i(u)|} - \frac{|N_i(w)|}{|N_i(u)|} - \frac{|N_i(w)|}{|N_i(v)|} + \frac{|N_i(w)|}{|N_i(u)|} + 1\\
&\geq \frac{|N_i(w)| - W(M_i^{max}(u,w))}{|N_i(u)|} + \frac{|N_i(w)| - W(M_i^{max}(v,w))}{|N_i(u)|}\\
&- \frac{|N_i(w)| - W(M_i(u,v))}{|N_i(u)|} + 1 - \frac{|N_i(w)|}{|N_i(v)|}\\
&\geq \frac{\sum_{(x,y,z)}(1-f(x,y)) + (1-f(z,y)) - (1-f(x,z))}{|N_i(u)|}\\
&+ 1 - \frac{|N_i(w)|}{|N_i(v)|}\\
&\geq 0
\end{aligned}
$$

where (x,y,z) denotes $(x,y) \in M_i^{max}(u,w), (z,y) \in M_i^{max}(v,w), (x,z) \in M_i(u,v)$. As a result, the triangle inequality is proved, and the theorem holds. □

4.3 Complexity Analysis

The time complexity of Algorithm 1 is mainly twofold. Firstly, it is to compute the k-neighborhood subgraphs for the two query nodes in line 3, which basically runs a bfs traversal from nodes u and v. Thus the complexity is $O(max(|G_k(u)|,|G_k(v)|)) = O(nd)$, where n is the number of nodes and d is the average degree.

Secondly, it is the `Matching` operation in line 10. Here we will analyze the maximum matching. The computation of StructSim-Max is dominated by the maximum matching operation. Hungarian algorithm [27] is the most efficient algorithm to do the maximum matching with $O(s^3)$ time complexity for an input size s. As a result, when we apply it to compute the maximum matching among the i-hop neighbors, the complexity is $O(|N_i|^3)$, where $|N_i|$ is the average size of the i-hop neighbors. We can adopt a greedy algorithm [5] as RoleSim to further improve the maximum matching time to $O(|N_i|^2 \log |N_i|)$.

As a whole, the time complexity of StructSim-Max for a single node pair is $O(nd + |N_i|^2 \log |N_i|)$. Note that $|N_i| = n$ in the worst case, which hinders its application to large real-world graphs. RoleSim, IcebergRoleSim, and NED suffer from such performance bottleneck as well. With the constraint of the algorithmic structures, there remains few spaces to further improve RoleSim (IcebergRoleSim) and NED. However, opportunities still present with the flexibility of the

StructSim framework in Algorithm 1. Our next section will reveal two techniques, namely BinCount matching and FMS-Index, that guarantee the admissibility of StructSim, and meanwhile make it efficient to compute.

5 Efficient StructSim Computation

5.1 BinCount Matching

We introduce the BinCount matching as a more efficient `Matching` alternative. Other than the performance gain, StructSim based on the BinCount matching still guarantees the admissibility of role similarity metric. In the following, we denote the BinCount-based StructSim as StructSim-BC.

Motivations. We first look into the following motivating example.

Example 2 Refer to Example 1. The maximum matching tends to match nodes with close degrees, which is reasonable by intuition. For example, node $x_2(2)$ matches $y_1(3)$ and $x_3(7)$ matches $y_2(7)$, where the number in the bracket is the degree. The remaining match $(x_1(1), y_3(9))$, however, looks like an outlier. Note that this "dissimilar" pair contributes very little to the result and can be simply ignored.

An alternative way of processing the matching may be: we group those i-hop neighbors into bins of ranges according to their degree, and we only match the nodes that belong to the same bin. The benefits of doing this are twofold. Firstly, the nodes in each resulted match are closer in degree by natural, analogous to that of the maximum matching. Secondly, searching is more local and hence more efficient.

Figure 4 depicts the above matching process for nodes in Example 1. The nodes in $N_i(u)$ and $N_i(v)$ are grouped into 4 bins, and only those in the same bins are matched. By doing so, x_2 still matches y_1 and x_3 matches y_2, same as the maximum matching. Nodes across the bins are not matched due to the small contribution to the similarity value.

We next show how we adjust the `Matching` and `Delta` operations in Algorithm 1 to implement the StructSim-BC algorithm.

Matching Operation. Motivated by Example 2, given two nodes u and v and their k-neighborhood subgraphs, we arrange the matching process in the i^{th} $(1 \le i \le k)$ level as: we group $N_i(u)$ and $N_i(v)$ into b bins according to the degrees, where the $j^{th}(1 \le j \le b)$ bins are denoted as $N_i^j(u)$ and $N_i^j(v)$ respectively, then we compute the matching $M_i(u,v)$ as

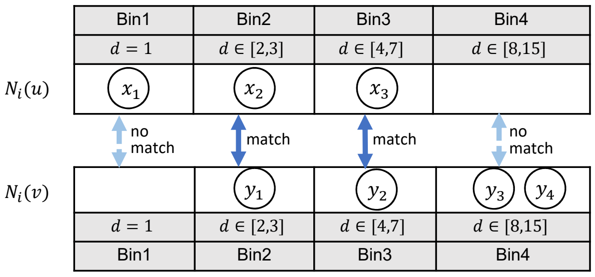


Fig. 4 The BinCount matching between $N_i(u)$ and $N_i(v)$

$$M_i(u,v) = \bigcup_{j=1}^{b} M(N_i^j(u), N_i^j(v)), \tag{3}$$

where $M(N_i^j(u), N_i^j(v))$ is a matching of the nodes in the j^{th} bins.

Intuitively, the smaller the node's degree is, the finer-grained the matching should be processed. It is not hard to see that two nodes of degrees 1 and 10 are less similar than two nodes of degrees 101 and 110, though with the same gap. As a result, we will arrange the bins based on the logarithmic value of the degree. Specifically, given a node $u \in V$ and its i-hop neighbors $N_i(u)$, let $b = 1 + \lfloor \log D \rfloor$, we will accordingly divide $N_i(u)$ into b bins, where the j^{th} bin $N_i^j(u)$ contains the nodes of degree in the range $[2^{j-1}, 2^j - 1]$. Given the bin arrangement, we define the BinCount set for $N_i(u)$ as

$$\mathcal{B}_i(u) = \{bc_i^1, bc_i^2, \dots, bc_i^b\}, \tag{4}$$

where $bc_i^j = |N_i^j(u)|$. Note that the BinCount sets of each level can be easily computed while constructing the k-neighborhood subgraph.

Definition 9 (BC-Index) For a node u, we define BC-Index as the data structure of the k-neighborhood subgraph $G_k(u)$, along with the BinCount set $\mathcal{B}_i(u)$ for $N_i(u)(1 \le i \le k)$.

Example 3 In Example 2, the observed maximum degree is $D = 10$, thus we configure $1 + \lfloor \log D \rfloor = 4$ bins for $N_i(u)$ and $N_i(v)$. Specifically, bin1 contains the nodes of $d = 1$, therefore $N_i^1(u) = \{x_1\}$ and $N_i^1(v) = \emptyset$. The nodes in the other bins are shown in Figure 4. The corresponding BinCount sets are $\mathcal{B}_i(u) = \{1, 1, 1, 0\}$ and $\mathcal{B}_i(v) = \{0, 1, 1, 2\}$, respectively.

Delta Operation. We look into a proper predefined similarity value of two matched nodes (x, y), namely $f(x, y)$ in line 11 of Algorithm 1. We agree with the maximum matching process that the degree ratio is a proper measurement of the predefined similarity of two nodes. Given the two nodes whose degree are within the range $[2^{j-1}, 2^j - 1]$ (i.e. they are in the same bin), it is obvious that the degree ratio is within $(\frac{1}{2}, 1]$, thus we may use any value in between for $f(x, y)$. As will be shown later, we assign $f(x, y) = 1$ to guarantee the admissibility of the algorithm. By doing so, we can accordingly simplify the Delta operation as

$$\varDelta_i(u, v) = \frac{\sum_{j=1}^{b} \min(bc_i^j(u), bc_i^j(v))}{\max(\sum_{j=1}^{b} bc_i^j(u), \sum_{j=1}^{b} bc_i^j(v))}, \quad (5)$$

where $bc_i^j(u) \in \mathcal{B}_i(u)$ and $bc_i^j(v) \in \mathcal{B}_i(v)$.

Example 4 Refer to Example 3, the corresponding BinCount sets at level i are $\mathcal{B}_i(u) = \{1, 1, 1, 0\}$ and $\mathcal{B}_i(v) = \{0, 1, 1, 2\}$, respectively. According to Equation 5, the $\varDelta_i(u, v)$ value based on the BinCount matching is $\varDelta_i(u, v) = \frac{2}{4} = 0.5$.

Admissibility. We prove the admissibility of StructSim-BC.

Theorem 4 (Admissibility) StructSim *with the* BinCount *matching is an admissible role similarity metric.*

Proof. We prove that for $\forall i \in [1, k]$, $\varDelta_i$ in Equation 5 satisfies all the properties in Definition 7.

Given the BinCount sets $\mathcal{B}_i(u) = [c_1, \cdots, c_b]$ for node u and $\mathcal{B}_i(v) = [c'_1, \cdots, c'_b]$ for node v, then we can get $\varDelta_i(u, v) = \frac{\sum_{j=1}^{b} min(c_j, c'_j)}{max(\sum_{j=1}^{b} c_j, \sum_{j=1}^{b} c'_j)}$ (Equation 5). Suppose that there are b nodes $x_1, x_2 \cdots x_b$ and all of them are with different node degree, i.e. $d_{x_1} \neq d_{x_2} \cdots \neq d_{x_b}$. Then, consider another two nodes p and q, and p has $\sum_{j=1}^{b} c_j$ neighbors, in which there are c_j copies for node x_j. Similarly, node q has $\sum_{j=1}^{b} c'_j$ neighbors, in which there are c'_j copies for node x_j. Note that if $u \equiv v$, $p \equiv q$ holds as well since the BinCount sets of nodes u and v at level i are the same. Based on the definition of RoleSim, we can get $\text{RoleSim}^1(p, q) = \frac{\sum_{j=1}^{b} min(c_j, c'_j)}{max(\sum_{j=1}^{b} c_j, \sum_{j=1}^{b} c'_j)}$, in which RoleSim is with the degree-binary initialization. Thus, $\varDelta_i(u, v) = \text{RoleSim}^1(p, q)$. As RoleSim has been proved to satisfy all the properties at any iteration [24], $\varDelta_i$ meets all the properties as well. As a result, StructSim-BC is an admissible role similarity metric. □

Remark 4 BinCount matching cannot be applied to RoleSim and NED. Note that the computation of the BinCount matching requires a predefined similarity function $f(x, y)$ in the Delta operation. While for RoleSim (or IcebergRoleSim), the similarity function varies in each iteration. As for NED, the similarity function at each level depends on the results of the previous level and is not possible to predefine either.

Complexity Analysis. The BinCount matching is very efficient, as it merely costs $O(k \log D)$ time to scan the BinCount set in Equation 5 at all levels. While the BinCount matching only accounts for one part of the time complexity of StructSim-BC, the other part remains to compute the BC-Index, which is equivalent to constructing the k-neighborhood subgraphs, and costs $O(nd)$ in the worst case (Section 4.3). Thus, the time complexity of StructSim-BC is $O(k \log D + nd)$.

Note that we can precompute the BC-Index for each node to improve the time complexity to $O(k \log D)$. An extra benefit of doing this is that we can trivially do the all-pair similarity computation using $O(n^2 k \log D)$ time, which is better than RoleSim by removing a d^2 term. More impressively, it is more space-efficient. Note that RoleSim must use $O(n^2)$ space to memorize the previous similarity values and NED needs $O(\tau^2)$ space to do the maximum matching shown in Table 1. On the other end, the BC-Index requires each node to maintain a k-level BinCount set, each of which is of size $O(\log D)$. As a whole, the index takes $O(kn \log D)$ space, while it allows the algorithm to compute the all-pair computation online without any memorization. The disadvantage of the precomputation is that it takes $O(n^2 d)$ time as a whole, which is cost-prohibitive to apply to large real-world graphs. To resolve this issue, we further propose the FM-sketch-based index, called FMS-Index, in the following section.

5.2 FM-Sketch-based Index Construction

We notice that the BinCount sets record count of the nodes in certain degree ranges, and each count can be approximated with a theoretical guarantee based on the well-known FM sketch [16]. This inspires us to propose the FMS-Index as an efficient substitution of BC-Index for StructSim-BC.

Flajolet-Martin Sketch. Flajolet-Martin sketch [16], denoted as FM sketch, was proposed to estimate the number of distinct elements in a multiset $\mathcal{S}$. Define $fm(\mathcal{S})$ as the FM sketch of $\mathcal{S}$, which is a binary number of l bits. Let $h : x \rightarrow [0, 2^l - 1]$, be a uniform hash function for $x \in \mathcal{S}$. We further define p_0 and p_1 as two position functions which give the positions of the left-most 0-bit and 1-bit of the binary number (starting from 0 and counting from left to right), respectively. For example, given a binary number 001010, $p_0 = 0$ and

$p_1 = 2$. We have

$$fm(\mathcal{S}) = \bigvee_{x \in \mathcal{S}} 1 << (l - p_1(h(x)) - 1), \tag{6}$$

where $\bigvee$ denotes a bitwise-or operation and $<<$ is the logical left shifting. Given $fm(\mathcal{S})$, we can estimate the number of distinct elements $z_{\mathcal{S}}$ as

$$\widetilde{z_{\mathcal{S}}} = \frac{1}{\phi} 2^{p_0(fm(\mathcal{S}))}, \tag{7}$$

where $\phi = 0.775351$ is a constant. To improve the accuracy, we can run r times of independent estimation, which results in the following theoretical guarantee.

Theorem 5 *For given $0 < \delta < 1$ and r, if $l = O(\log z_{\mathcal{S}} + \log r + \log \delta^{-1})$, then $|z_{\mathcal{S}} - \widetilde{z_{\mathcal{S}}}| < \epsilon z_{\mathcal{S}}$ holds with the probability at least $1-\delta$, where $\epsilon = O(\sqrt{\frac{\log \delta^{-1}}{r}})$ [16, 32].*

FM-Sketch-Based Index Construction. Given a node u, we show how to utilize the FM sketch to estimate the BinCount set $\mathcal{B}_i(u)$ of $N_i(u)$. Note that this is non-trivial, as we do not have $N_i(u)$ at hand to apply the FM sketch. To resolve this, we use the fact that

$$N_i(u) = N_{\le i}(u) \setminus N_{\le i-1}(u), i \ge 1,$$

where $N_{\le i}(u)$ is the i-hop reachable neighbors (Definition 2). Analogous to Equation 4, we can compute the BinCount set $\mathcal{B}_{\le i}(u)$ for $N_{\le i}(u)$ using the same bin arrangement. It is immediate that

$$bc_i^j(u) = bc_{\le i}^j(u) - bc_{\le i-1}^j(u), i \ge 1, \forall 1 \le j \le 1 + \lfloor \log D \rfloor,$$

where $bc_i^j(u)$ and $bc_{\le i}^j(u)$ are the j^{th} element of $\mathcal{B}_i(u)$ and $\mathcal{B}_{\le i}(u)$, respectively. The idea emerges to estimate $bc_{\le i}^j(u)$ in the above equation instead of directly touching $bc_i^j(u)$.

To do so, we compute the FM sketch $fm(N_{\le i}^j(u))$, and according to Equation 7, we have an estimation of $bc_{\le i}^j(u)$ as

$$\widetilde{bc}_{\le i}^j(u) = \frac{1}{\phi} 2^{p_0(fm(N_{\le i}^j(u)))} \tag{8}$$

We then solve the last puzzle of how to efficiently compute $fm(N_{\le i}^j(u))$. We first cite the following lemma [32].

Lemma 1 *Given two multisets $\mathcal{S}_1$ and $\mathcal{S}_2$, we can get $fm(\mathcal{S}_1 \bigcup \mathcal{S}_2) = fm(\mathcal{S}_1) \bigvee fm(\mathcal{S}_2)$, where $fm(\mathcal{S}_1)$ and $fm(\mathcal{S}_2)$ are the FM sketches for $\mathcal{S}_1$ and $\mathcal{S}_2$ respectively.*

It is obvious that $N_{\le i}(u) = \bigcup_{v \in N(u)} N_{\le (i-1)}(v), i \ge 2$, and with Lemma 1, it is not hard to see

$$fm(N_{\le i}^j(u)) = \bigvee_{v \in N(u)} fm(N_{\le (i-1)}^j(v)), i \ge 2 \tag{9}$$

Note that the cases of $i = 0$ and $i = 1$ for Equation 9 are easy to compute based on $N_0(u)$ and $N(u)$. With Equation 8 and Equation 9, we will estimate $bc_i^j(u)$ for $\forall i \ge 1$ as

$$\widetilde{bc}_i^j(u) = \widetilde{bc}_{\le i}^j(u) - \widetilde{bc}_{\le i-1}^j(u), 1 \le j \le 1 + \lfloor \log D \rfloor \tag{10}$$

Lemma 2 *Suppose that the distribution of nodes in the bins is the same for each level of the k-neighborhood subgraph, then Equation 10 has the approximate ratio $\frac{a+b}{a-b}\epsilon$ with confidence $1 - \delta$ if the FM sketch has length $l = O(\log n + \log r + \log \delta^{-1})$, where $\epsilon = O(\sqrt{\frac{\log \delta^{-1}}{r}})$, $n = |V|$, $a = |N_{\le i}(u)|$ and $b = |N_{\le (i-1)}(u)|$.*

Proof. Denote x, y, z as the true values of $bc_{\le i}^j(u)$, $bc_{\le i-1}^j(u)$ and $bc_i^j(u)$ respectively, and x', y', z' as the estimated ones, then $z = x - y$ and $z' = x' - y'$. According to Theorem 5, we have $x - \epsilon x < x' < x + \epsilon x$ and $y - \epsilon y < y' < y + \epsilon y$. Thus, $\frac{|z'-z|}{z} < \epsilon \frac{x+y}{x-y}$. Based on the assumption, the lemma holds. □

Remark 5 In Lemma 2, when $a \gg b$, the approximate ratio approaches ϵ. Note that this is often true in real-world graphs for $i \le 6$, where the number of i-hop reachable neighbors often increases exponentially. We confirm this for real-world datasets in Section 6.3. It is worth noting that in Algorithm 1, we prefer to set $w_i < \frac{1}{i+1}$ to make the contribution of each layer gradually decreases with i, and in this sense, such approximation error is negligibly small for $i > 6$.

Next, we give the pseudo-code for FMS-Index construction, shown in Algorithm 2, which can be trivially performed r times to improve the accuracy as shown in Lemma 2.

Complexity Analysis. The FMS-Index now corresponds to computing Equation 9 for each node $u \in V$ in all levels $1 \le i \le k$ and all bins $1 \le j \le 1 + \lfloor \log D \rfloor$. We can apply the dynamic programming while using Equation 9 as the transaction function. The time complexity of building the FMS-Index is now $O(rknd \log D)$, where r is the number of times to perform FM sketch and the term nd comes from accessing the neighbors from each node. This is evidently better than the $O(n^2 d)$ time complexity of BC-Index.

Parallelization. In Algorithm 2, the most costly part is to iterate over all nodes in the graph to compute the

Algorithm 2: FM-Sketch-Based Index Construction

```
Input  : A data graph $G$, parameter $k$.
Output : FMS-Index.
1  $b \leftarrow 1 + \lfloor \log D \rfloor$;
2  foreach node $u \in V$ do
3      foreach bin $j \in [1, b]$ do
4          Calculate $fm(N^j_{\leq 0}(u))$, $fm(N^j_{\leq 1}(u))$ based
           on $\{u\}$ and $N(u) \bigcup \{u\}$, respectively;
5  foreach level $i \in [2, k]$ do
6      foreach node $u \in V$ do
7          foreach bin $j \in [1, b]$ do
8              $fm(N^j_{\leq i}(u)) \leftarrow 0$;
9              foreach node $w \in N(u)$ do
10                 $fm(N^j_{\leq i}(u)) \leftarrow$
                     $fm(N^j_{\leq i}(u)) \bigvee fm(N^j_{\leq i-1}(w))$;
11 Calculate FMS-Index based on Equation 8 and
   Equation 10;
12 return FMS-Index.
```

sketches, and it may be the bottleneck while handling large graphs. Fortunately, this can be easily paralleled as the computation of each node is only relied on the sketches of its neighbors at the previous level and is obviously independent. We simply round-robin the nodes to distribute the loads to all available threads, which already achieves a satisfactory scalability trend in the experiment (Figure 13(b)). There are other distribution strategies [2] that may result in better load balance. As this is not the main focus of this paper, we leave it as interesting future work.

RoleSim can use a similar idea to parallelize, but its main problem is the $O(n^2)$ space occupation, which makes it almost infeasible for large graphs. As IcebergRoleSim only optimizes RoleSim heuristically, it still occupies $O(n^2)$ space in the worst case. Similarly, NED is not capable of adopting parallelism as well since it needs $O(\tau^2)$ space to perform maximum matching, and τ cannot be bounded by the number of nodes in the graph.

6 Experimental Evaluation

In this section, we conduct extensive experiments to compare StructSim with RoleSim, IcebergRoleSim, and NED in terms of effectiveness and efficiency. The authors kindly offer the codes for RoleSim, IcebergRoleSim, and NED. RoleSim and IcebergRoleSim were implemented in C++. NED was originally implemented in Java, while we re-implement it in C++ for a fair comparison. We implement StructSim and all its variants in C++. The experiments are by default conducted on a machine configured with an Intel(R) Xeon(R) CPU E3-1220 v6 @ 3.00GHz and 64GB memory. As for the experiments of parallelism, we use a server of 2 Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20GHz (each has 20 cores 40 threads) and 512GB memory.

6.1 Experimental Settings

Algorithms. We summarize the algorithms and their parameters in the experiments as follows.

- RoleSim: An iterative algorithm proposed in [24]. β is set to 0.9 and the convergence is defined as when the values change by less than 1% on previous values, same as [24].
- IcebergRoleSim: An extension for RoleSim in [25]. θ is set to 0.8. β and convergence are defined the same as RoleSim.
- NED: A hierarchy-based algorithm proposed in [60]. The number of levels k is set to 3, as suggested by the authors.
- StructSim-Max: The StructSim based on maximum matching in Section 4.2. We set the number of levels $k = 10$. We let $\omega_i = \frac{df}{i+1}$, where i is the level and $0 < df \leq 1$ is a damping factor. We set $df = 0.8$, i.e., $\omega_i = \frac{0.8}{i+1}$ by default.
- StructSim-BC: The BinCount-based StructSim in Section 5.1. The parameters are the same as StructSim-Max. We set r to 150 for FM sketch. We further configure the following two variants:
  - StructSim-BC^b: StructSim-BC with BC-Index.
  - StructSim-BC^f: StructSim-BC with FMS-Index.

6.2 Effectiveness on Case Studies

Here, we evaluate the effectiveness of StructSim against RoleSim and NED. In addition to the existing role similarity measures (i.e., RoleSim, IcebergRoleSim, and NED), we include two state-of-the-art embedding-based algorithms, namely GraphWave [15] and struct2vec [41], as another two baselines to investigate the effectiveness of the proposed StructSim. Specifically, both GraphWave [15] and struct2vec [41] compute the structural node embedding, which is a representation of node encoding its structural information. We compute the Euclidean distance of two nodes' embeddings and use the same normalization as NED to compute similarity values. The codes of GraphWave and struct2vec are provided by the respective authors, and we use the default settings as advised in the related papers.

The effectiveness of different algorithms is evaluated via four case studies, which have clear roles as ground

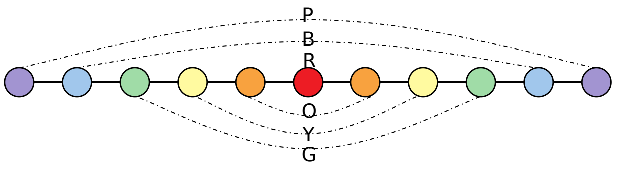


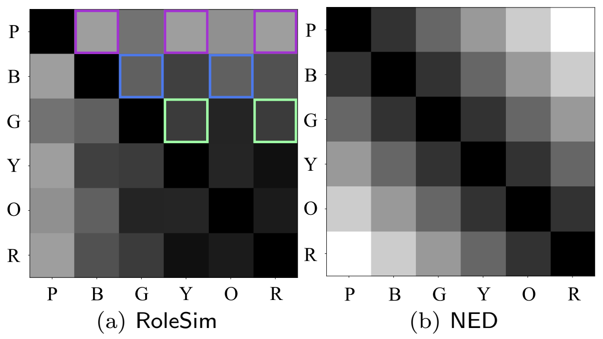
Fig. 5 An even path graph $\mathcal{P}$. Different colour indicates different roles.

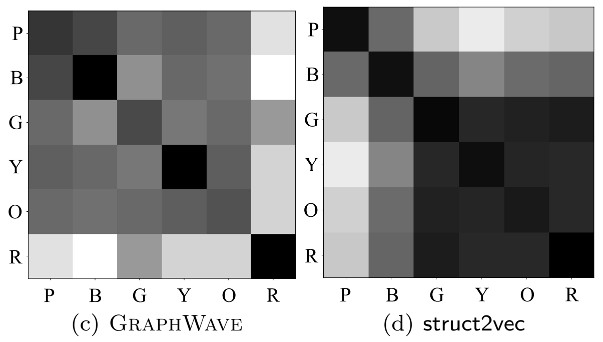
truth to judge each algorithm's quality. Specifically, we first use an even-path graph to demonstrate the "odd-distance abnormality" of RoleSim (Theorem 1). Further, we extend the even-path graph to a Barbell graph, which shows that RoleSim performs the worst on rendering the similarity rankings, and the culprit may be the aforementioned "odd-distance abnormality". Finally, we adopt two kinds of real-life networks, i.e., DBLP co-authorship network and air-traffic networks to investigate the performance of different algorithms in tackling the tasks of ranking, clustering, and classification.

Even Path. In Figure 5, we show an even path graph $\mathcal{P}$ of length 10. Clearly, $\mathcal{P}$ has 6 equivalence classes denoted by different node colours and notations that are used as their **roles**.

In the heat maps shown in Figure 6, we present the similarity values given by different algorithms. Intuitively, we expect that closer nodes in Figure 5 should be more similar in their roles. For example, we expect the role similarity metric rendering the ranking $\mathsf{Sim}(P,P) > \mathsf{Sim}(P,B) > \cdots > \mathsf{Sim}(P,R)$ for the "P" nodes, and $\mathsf{Sim}(B,B) > \mathsf{Sim}(B,G) > \cdots > \mathsf{Sim}(B,R)$ for the "B" nodes, and so forth. IcebergRoleSim is omitted since its results are similar to RoleSim. In the heat map, we cross compare all pairs of node classes in a 6x6 grid, and the darker the block is, the larger the similarity value is given by the algorithm.

According to Figure 6, the diagonal blocks in all heat maps of role similarity measures (i.e., RoleSim, NED, StructSim-Max and StructSim-BC) are the darkest, indicating that RoleSim, NED and our StructSim can guarantee *automorphism confirmation* of role similarity metric (Definition 7). In contrast, the embedding-based algorithms, namely GraphWave and struct2vec, cannot ensure that automorphic nodes have identical embeddings, and thus both of them fail to satisfy the property of *automorphism confirmation*. From Figure 6(a), we can observe that RoleSim always renders the rankings of $\mathsf{RoleSim}(P,B) = \mathsf{RoleSim}(P,Y) = \mathsf{RoleSim}(P,R)$, $\mathsf{RoleSim}(B,G) = \mathsf{RoleSim}(B,O)$ and $\mathsf{RoleSim}(G,Y) = \mathsf{RoleSim}(G,R)$, as indicated by the outlined blocks. This observation reveals the "odd-distance abnormality" of RoleSim and conforms with





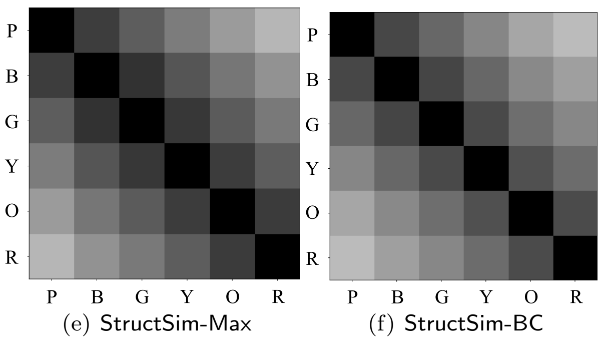


Fig. 6 Heat maps of different algorithms on computing the even path $\mathcal{P}$. Outlined blocks in RoleSim indicates "odd-distance abnormality".

the theoretical analysis in Theorem 1. Note that among all the evaluated algorithms, only NED and our StructSim (both StructSim-Max and StructSim-BC) give reasonable results that follow the expected rankings.

Barbell Graph. To show that such "odd-distance abnormality" of RoleSim may present on a broader basis, we conduct the case study on a Barbell graph, a regular extension to the path graph. A Barbell graph $B(p,q)$ is formed by connecting two p-cliques using a path of length q. In Figure 7, we show a Barbell graph $B(10,10)$, which contains an even path of length 10. The Barbell graph has 7 equivalent classes indicated by different colors (and notations), which stand for their **roles**.

Figure 8 shows the heat maps given by different algorithms on the Barbell graph $B(10,10)$. Analogous to

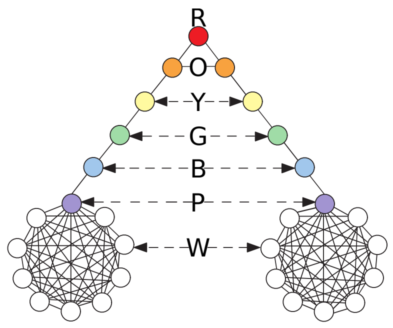
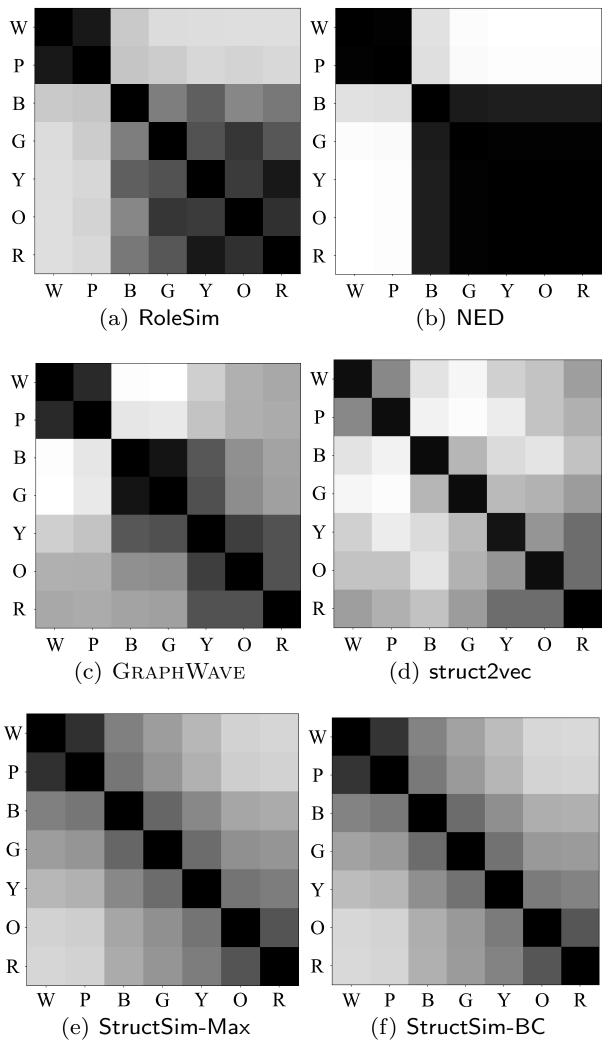


Fig. 7 The graph structure of the Barbell graph $B(10, 10)$



(a) RoleSim (b) NED (c) GraphWave (d) struct2vec (e) StructSim-Max (f) StructSim-BC

Fig. 8 Heat maps of different algorithms on computing $B(10, 10)$

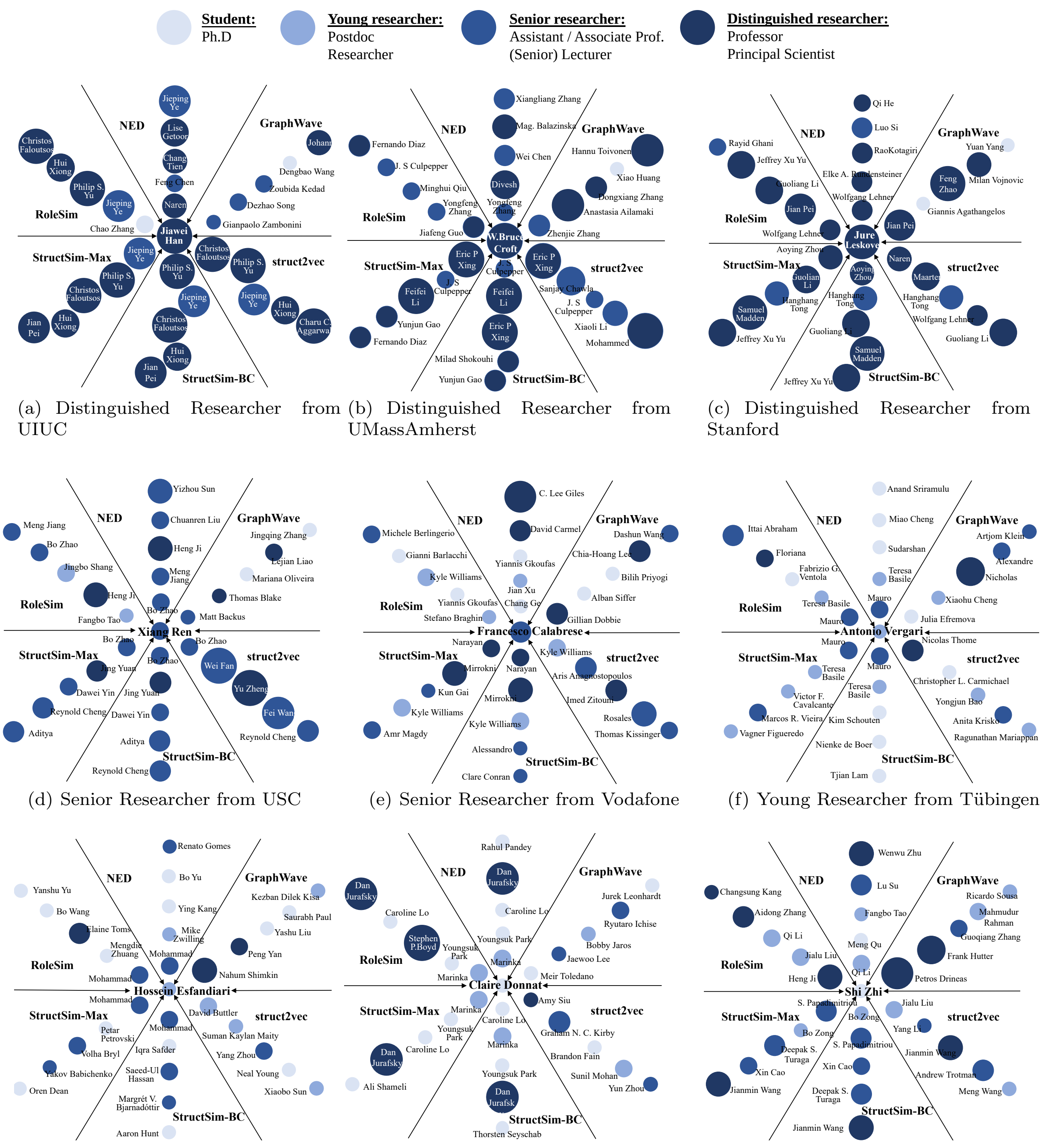
the case of even path, we also expect the similarity rankings, such as $\mathsf{Sim}(P,P) > \mathsf{Sim}(P,B) > \cdots > \mathsf{Sim}(P,R)$ for the "P" nodes, and $\mathsf{Sim}(B,B) > \mathsf{Sim}(B,G) > \cdots > \mathsf{Sim}(B,R)$ for the "B" nodes, and so forth. As observed from Figure 8, RoleSim is still messed up with the similarity rankings as that in Figure 6, which aligns with our speculation of "odd-distance abnormality". It is worth noting that the ranking of NED on computing the Barbell graph can hardly be distinguished from the heat map, as shown in Figure 8(b). One possible reason for this is that NED generates k-adjacent trees by involving the already visited nodes, and the tree size (and thus the edit distance) distributes extremely. In terms of the embedding-based algorithms, i.e., GraphWave and struct2vec, they perform better than NED in generating more distinguishable scores, but they still fail to give results aligned with the expected rankings. Among all the algorithms, only StructSim-Max and StructSim-BC render the heat maps according to the expected rankings.

Co-authorship Graph. We then evaluate the performance of different algorithms on the task of relevance ranking. We adopt a co-authorship graph extracted from DBLP[4], which includes authors who published at least one paper on six top "DB/DM/IR" conferences (SIGMOD, VLDB, ICDE, KDD, SIGIR, and WWW) from 2014 to 2018. Instead of using the full graph, we extract a subset of the DBLP dataset as [24] due to the high computational cost of RoleSim. In summary, this graph contains 18,587 nodes and 55,783 edges.

As most researchers' positions are available online, we can use the position to indicate the role, as shown in Figure 9. We categorize the positions into four roles: "Student", "Young researcher", "Senior researcher" and "Distinguished researcher", which are indicated with colours from light to dark, and the detailed positions regarding each category are also listed. Note that the position itself may not fully reflect the role. For example, a "Young researcher" from a laureate research organization may act like a "Senior researcher". Therefore, we also include the H-index as an auxiliary indicator of the role. H-index is defined as such an h value that at least h papers of the researcher have been cited at least h times, which is an important indicator of the researcher's impact and can be obtained from google scholar. We consider 7 ranges of H-index: $[0, 10), [10, 20), \ldots, [50, 60), [60, \infty)$, and present the H-index range of each researcher in Figure 9 as proportional to the radius of the corresponding circle.

Figure 9 presents the top-5 most similar results given by evaluated algorithms for 9 query researchers

[4] `https://dblp.uni-trier.de/xml/`



(a) Distinguished Researcher from UIUC (b) Distinguished Researcher from UMassAmherst (c) Distinguished Researcher from Stanford

(d) Senior Researcher from USC (e) Senior Researcher from Vodafone (f) Young Researcher from Tübingen

(g) Young Researcher from Google (h) Student from Stanford (i) Student from UIUC

Fig. 9 Case study on the co-authorship graph: the top-5 most similar results of given researchers.

with all the four roles. Note that nodes with similar roles should have close color and radius in Figure 9. The results of IcebergRoleSim and StructSim-BC^f are close to RoleSim and StructSim-BC^{b}respectively, and thus are omitted in Figure 9 for clear presentation. For simplicity, in the following, we only include the performance analysis regarding the roles of “Distinguished researcher” and “Senior researcher”, which show that our StructSim as a whole performs better than the other algorithms in terms of returning more reasonable results. The results for the other roles are similar and hence omitted.

For the role “Distinguished researcher” in Figure 9(a) and Figure 9(b), StructSim and struct2vec performs much better considering the similar positions and H-index ranges of the returned researchers. In comparison, RoleSim, NED, and GraphWave all return less reasonable results. Specifically, in Figure 9(a), RoleSim returns a much less influential “Student” as the top-1 result for a “Distinguished researcher” with big impact, and NED and GraphWave return much less influential researchers than the subject. In Figure 9(c), StructSim, RoleSim, NED and struct2vec give similar results regarding research positions. However, when taking H-index into account, StructSim and struct2vec return researchers with higher H-index ranges than the other algorithms. GraphWave, however, gives unpromising results, with two less influential “Ph.D” students in top-5 results for a “Distinguished researcher”.

As for the role “Senior researcher” in Figure 9(d), StructSim and NED perform the best as they give the most results with similar positions and close H-index ranges, while the other algorithms perform not well either in position (e.g., RoleSim and GraphWave) or in H-index (e.g., struct2vec). In Figure 9(e), struct2vec gives the most reasonable results, followed by StructSim and GraphWave. RoleSim and NED do not perform well in this case as the results differ significantly in both position and H-index from the query researcher.

To deliver a more comprehensive analysis, we compute the top-5 most similar results for 20 randomly selected researchers, and the number of researchers of each role distributes evenly. We then evaluate the effectiveness of the algorithms using the distance metrics of position and H-index (for accuracy), and the Normalized Discounted Cumulative Gain (short as nDCG, for ranking quality [14]). Intuitively, these metrics evaluate how close the computed similar results are to the subject regarding their positions and H-index values. Next, we first introduce several basic definitions of the metrics, then present the related results in Table 3.

We define $P(x)$ and $H(x)$, which return the level of position and the range of H-index for the researcher x, respectively. Specifically, there are total 4 position levels from 1 to 4 indicating positions from “Student” to “Distinguished professor”, and 7 ranges of H-index from 1 to 7 as presented before. Thereafter, the position distance (resp. the H-index distance) between two nodes is defined as the absolute difference between their $P(\cdot)$ (resp. $H(\cdot)$) values. The position and H-index distances are denoted as $dist_P$ and $dist_H$, respectively. We also give each returned researcher the relevance score in terms of position and H-index, where each of them has three levels: 2-very relevant ($dist = 0$), 1-some relevant ($dist = 1$) and 0-non relevant ($dist > 1$). We then label each returned researcher with a relevance score as the average of those of position and H-index.

Example 5 Suppose there are two researchers x_1 and x_2. x_1 is a professor with H-index 65, and x_2 is a principal scientist with H-index 57. Accordingly, we have $P(x_1) = 4$, $H(x_1) = 7$ and $P(x_2) = 4$, $H(x_2) = 6$. The position distance between x_1 and x_2 is $dist_P(x_1, x_2) = |4-4| = 0$, and the H-index distance is $dist_H(x_1, x_2) = |7-6| = 1$. Thus, the relevance scores between x_1 and x_2 in terms of position and H-index are 2 and 1, respectively. As a result, the relevance score between researchers x_1 and x_2 is 1.5.

Ideally, two researchers with the same role should have the same research position and range of H-index, and thus get 0 for both position distance and H-index distance. We then use the average and standard deviation values of the absolute differences between the returned results and the query researchers, denoted as d_A and d_S respectively. d_A and d_S are calculated as follows,

$$d_A = \frac{\sum_{x\in C} d(x)}{|C|} \tag{11}$$

$$d_S = \sqrt{\frac{\sum_{x\in C}(d(x) - d_A)^2}{|C| - 1}} \tag{12}$$

where C is a set including the 20 researchers in this paper. $d(x)$ is the total distance for each query researcher x, namely $d(x) = \sum_{i=1}^{5} dist(x, R_x[i])$, in which R_x is the node set including the top-5 similar results for node x and $N_x[i]$ is the i^{th} element in R_x.

Table 3 lists the results of all the evaluated algorithms. Note that for nDCG, the larger, the better, while for d_A and d_S, the smaller, the better. We can observe that the StructSim variants (StructSim-Max, StructSim-BC^b, and StructSim-BC^f) give the best ranking quality, as the nDCG value is larger, followed by

Table 3 Ranking quality of different algorithms on Co-authorship graph. SS is short for StructSim. PD is short for position distance, and HD is short for H-index distance.

Metric	RoleSim	IcebergRoleSim	NED	GraphWave	struct2vec	SS-Max	SS-BC^b	SS-BC^f
nDCG	0.40	0.43	0.47	0.42	0.40	**0.48**	**0.48**	**0.48**
d_A (PD)	5.65	5.50	4.45	6.10	5.50	**4.15**	4.30	4.60
d_s (PD)	3.22	3.28	3.35	**2.20**	3.97	3.50	3.40	3.28
d_A (HD)	6.45	5.85	6.70	6.65	6.75	5.30	**5.25**	6.25
d_s (HD)	5.27	4.92	6.04	6.12	4.30	**3.50**	3.70	5.44

NED, IcebergRoleSim, and GraphWave. For the distance metrics, other than d_S of the position distance, the variants of StructSim clearly outperform the other baseline algorithms as they give the smallest distance values. It is worth noting that StructSim-BC^b(based on the exact BC−Index) and StructSim-BC^f(based on FMS−Index) perform comparatively with StructSim-Max regarding effectiveness in this case study, which indicates that the techniques of BinCount matching and FM-sketched-based indexing will not affect the effectiveness of the algorithm.

Real-World Air-Traffic Networks. Here, we evaluate the effectiveness of different algorithms on the clustering and classification tasks. We use three real-world benchmark datasets, i.e., Brazilian air-traffic network (Brazil)[5], European air-traffic network (Europe)[6] and American air-traffic network (USA)[7] according to [41]. Every node in these datasets has a label related to its topology [41], and thus we can use the labels as the ground truth of roles. In summary, Brazil contains 131 nodes, 1,038 edges and 4 labels. Europe contains 399 nodes, 5,995 edges and 4 labels. USA contains 1,190 nodes, 13,599 edges and 4 labels, respectively.

We perform the clustering and classification tasks on each dataset. The settings of each task are shown as follows: (1) for the clustering task, we apply k-means on the similarity scores computed by each algorithm and set the number of clusters to 4 as each dataset contains 4 labels. Accordingly, we use evaluation metrics of Normalized Mutual Information (NMI) [45], Homogeneity [42], and Completeness [42]. To be specific, NMI [45] measures the agreement of the computed clustering assignment versus the ground-truth assignment. Homogeneity [42] computes the conditional entropy of the computed clustering assignment given the ground-truth assignment. Completeness [42] calculates how many nodes within the same ground-truth cluster are assigned to the same cluster; and (2) for the classification task, we use k-NN algorithm to predict the label of each node based on its 5-nearest neighbors [8] (i.e., top-5 similar nodes to the target node). We then adopt the metrics of accuracy and F1-score to evaluate classification performance. *Note that the values of the metrics for both clustering and classification are the higher, the better.*

Table 4 reports the results of each algorithm for clustering and classification. Note that NED cannot finish the all-pair computation in two days for Europe and USA, denoted as "-" in Table 4. Specifically, we observe that the size of certain 3-adjacent trees of these two datasets exceeds 100, 000 nodes, and it fails naturally because of the cubic complexity. According to Table 4, StructSim variants get the most highest values on the evaluation metrics across different datasets, and thus perform better than both role similarity measures (i.e., RoleSim, IcebergRoleSim and NED) and embedding-based algorithms (i.e., GraphWaveand struct2vec). Additionally, as observed in the previous case study, StructSim-BC^{b}and StructSim-BC^{f}perform similarly to StructSim-Max, while they are much more efficient to compute as will be later evaluated.

Analysis of Parameters. According to Algorithm 1, there are two parameters, i.e., k and w_i for StructSim computation, in which k is the number of levels considered in k-hop neighborhood subgraph, w_i is a weighting parameter computed by $w_i = \frac{df}{i+1}$, and $0 < df \leq 1$ is a damping factor. We next study how k and damping factor df influence the performance of StructSim.

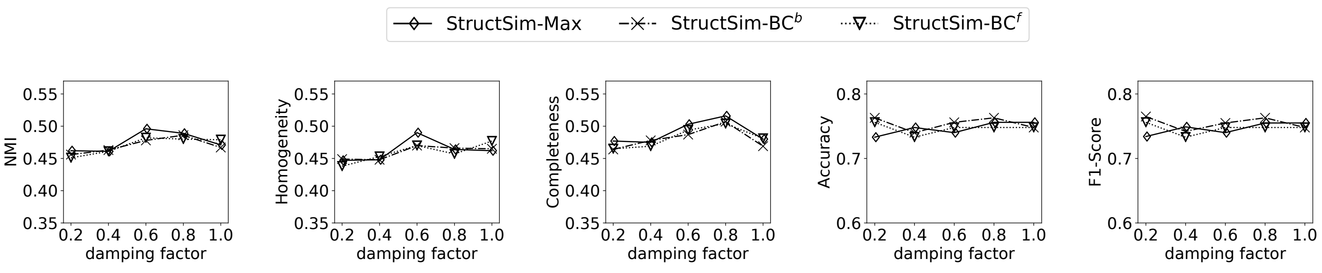
Figure 10 reports the clustering and classification results of StructSim variants on the Brazil dataset while varying k and damping factor. Specifically, we vary the damping factor from 0.2 to 1 (k is fixed to 4, the graph diameter) and vary k from 1 to 4 (damping factor is set to 0.8). According to Figure 10(a), we can observe that in the tasks of clustering and classification, StructSim variants achieve the best performance at different damping factor values, and df = 0.6 - 0.8 is, in general, a good setting for both tasks. Intuitively, a relatively large damping factor makes the weighing factors

[5] http://www.anac.gov.br/
[6] http://transtats.bts.gov/
[7] http://ec.europa.eu/

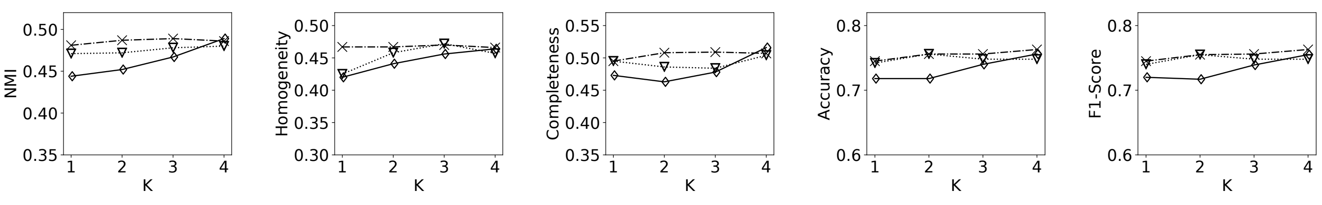
[8] We tried $k = 1$, $k = 5$ and $k = 8$ for k-NN, and adopt $k = 5$ as all the baselines get better results under this k value.

Table 4 Clustering and classification performance of different algorithms on Air-Traffic networks. SS indicatesStructSim.

Dataset	Metric	RoleSim	IcebergRoleSim	NED	GraphWave	struct2vec	SS-Max	SS-BC^b	SS-BC^f
Brazil	NMI	0.476	0.486	0.478	0.120	0.388	**0.489**	0.486	0.480
	Homogeneity	0.460	0.470	**0.478**	0.110	0.367	0.464	0.466	0.457
	Completeness	0.492	0.502	0.479	0.131	0.411	**0.516**	0.507	0.504
	Accuracy	0.710	0.740	0.733	0.504	0.702	0.756	**0.763**	0.748
	F1-Score	0.709	0.746	0.732	0.504	0.704	0.754	**0.763**	0.748
Europe	NMI	0.311	0.303	-	0.011	0.300	**0.339**	0.299	0.331
	Homogeneity	0.309	0.299	-	0.010	0.281	**0.336**	0.291	0.329
	Completeness	0.313	0.306	-	0.011	0.320	**0.341**	0.308	0.334
	Accuracy	0.589	**0.612**	-	0.381	0.586	0.581	0.591	0.574
	F1-Score	0.59	**0.614**	-	0.367	0.583	0.58	0.59	0.572
USA	NMI	0.299	0.306	-	0.008	0.224	0.287	**0.309**	0.288
	Homogeneity	0.288	**0.302**	-	0.007	0.206	0.282	0.299	0.283
	Completeness	0.311	0.310	-	0.009	0.244	0.291	**0.319**	0.293
	Accuracy	**0.670**	0.650	-	0.306	0.619	0.639	0.647	0.634
	F1-Score	**0.664**	0.645	-	0.299	0.613	0.634	0.64	0.628



(a) varying damping factor *df*



(b) varying K

Fig. 10 Clustering and classification performance of StructSim on the Brazil dataset while varying k and decay factor

decrease smoothly with respect to the number of levels, and thus far neighbors can still contribute to the computation. Based on Figure 10(b), we can observe that the performance of StructSim variants generally gets better when k becomes larger. This is reasonable as larger k involves more neighbors and thus contains more structural information. As a result, $k = diameter$ of the graph is a reasonable setting. Note that both k and damping factor can also be set by users or be tuned with training data for a specific application.

As discussed in Remark 2, in addition to the degree ratio initialization (line 1 of Algorithm 1), our StructSim framework also admits the degree-binary initialization. In the following, we study the performance of our StructSim on the clustering and classification tasks with different initialization functions, i.e., degree-binary (DB) and degree-ratio (DR). Note that RoleSim and IcebergRoleSim can be initialized by DB and DR as well, and thus we include them for comparison. For NED, GraphWave, and struct2vec, they are free from initialization functions, and we omit their results accordingly.

Table 5 reports the results of each algorithm with different initialization functions when performing clustering and classification tasks on the Brazil dataset. We can observe that RoleSim, IcebergRoleSim,

Table 5 Clustering and classification performance on Brazil dataset when using different initialization functions. DB and DR denotes degree-binary and degree-ratio initializations respectively.

Metric	Initialization	RoleSim	IcebergRoleSim	StructSim-Max	StructSim-BC^b	StructSim-BC^f
NMI	DB	0.437	0.462	0.437	0.486	0.480
	DR	0.476	0.486	0.489	0.486	0.480
Homogeneity	DB	0.428	0.447	0.424	0.469	0.456
	DR	0.460	0.470	0.464	0.466	0.457
Completeness	DB	0.447	0.477	0.450	0.503	0.503
	DR	0.492	0.502	0.516	0.507	0.504
Accuracy	DB	0.720	0.725	0.754	0.763	0.748
	DR	0.710	0.740	0.756	0.763	0.748
F1-Score	DB	0.720	0.724	0.752	0.763	0.748
	DR	0.709	0.746	0.754	0.763	0.748

and StructSim-Max give better performances with the degree-ratio initialization. This is reasonable as degree-ratio gives initial scores in a more fine-grained manner, while degree-binary gives score 0 for all the node pairs with different degree values even though some pairs are close in degree. It is worth noting that with the technique of BinCount matching, both StructSim-BC^{b}and StructSim-BC^{f}are not sensitive to initialization functions. This is because the computation of $\varDelta$ values for these algorithms does not rely on the initialization function (Equation 5).

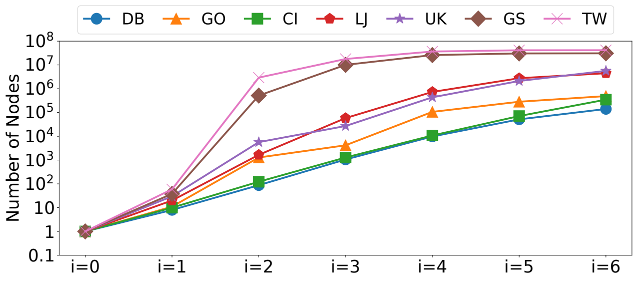


Fig. 11 The average number of i-hop reachable neighbors for each dataset

6.3 Efficiency

Datasets. We use 11 publicly available real-world datasets, in which Blog is from [1,49] and the others are downloaded from SNAP [30] and Webgraph [7]. These graphs are preprocessed as unlabelled and undirected simple graphs. Table 6 lists the statistics of each dataset. Column name gives the abbreviation to represent each dataset in the figures; n and m report the number of nodes and the number of edges, respectively; d and D show the average and maximum node degree in each dataset; and the index size is the size of FMS-Index for StructSim-BC.

Analysis of i-Hop Reachable Neighbors. Recall Lemma 2, when $a \gg b$, the approximate ratio approaches ϵ with confidence $1 - \delta$, in which a and b denote the number of i-hop reachable neighbors and the number of $(i-1)$-hop reachable neighbors, respectively. In order to verify the assumption ($a \gg b$), we randomly sample 10,000 nodes for each dataset, and calculate the average number of i-hop reachable neighbors for $0 \leq i \leq 6$, shown in Figure 11. We omit the first four smaller datasets as BFS-Index can be rapidly constructed for them.

According to Figure 11, one can find that the number of i-hop reachable neighbors almost increases exponentially with respect to i for i from 0 to 6 for the datasets DB, CI, LJ, and UK. As for the datasets GO, GS, and TW, exponential increasing happens within the 4-hop reachable neighbors as well.

Performance of Index Construction. Figure 12 presents the index construction time of BC-Index and FMS-Index for each dataset and INF is denoted for the cases that cannot terminate within two days.

According to Figure 12, it is clear that the FMS-Index is significantly faster than the BC-Index with up to two orders of magnitude. Specifically, it runs out of time to construct the BC-Index for the CI dataset with less than 17 million edges, while the FMS-Index can terminate on the dataset UK with more than 260 million edges. One outlier is the BL dataset where the FMS-Index is slower than BC-Index, and this can be explained based on the time complexity of BC-Index ($O(n^2d)$) and FMS-Index ($O(rknd \log D)$): when n is small, $rk \log D$ may be larger than n. In Figure 12, we omit the results of GS and TW as they run out of time to build both BC-Index and FMS-Index. Next, we will evaluate how the parallelization can boost the

Table 6 Graph Statistics

Datasets	Name	n	m	d	D	Index Size
Blog	BL	10,312	333,983	64.78	3,992	2.3 MB
Astroph	AS	18,771	198,050	21.10	504	5.3 MB
Enron	EN	36,692	183,831	10.02	1,383	10.8 MB
Epinions	EP	75,879	405,740	10.69	3,044	29.0 MB
DBLP	DB	317,080	1,049,866	6.62	343	108.5 MB
Google	GO	875,713	4,322,051	9.87	6,332	426.0 MB
Cit	CI	3,774,768	16,518,947	8.75	793	1.4 GB
LiveJournal	LJ	5,203,763	48,709,621	18.72	15,016	2.5 GB
UK	UK	18,483,186	261,787,258	28.33	194,955	12.4 GB
Gsh	GS	30,809,122	489,675,683	31.79	2,175,983	17.7 GB
Twitter	TW	41,652,230	1,202,513,046	57.74	2,997,487	27.1 GB

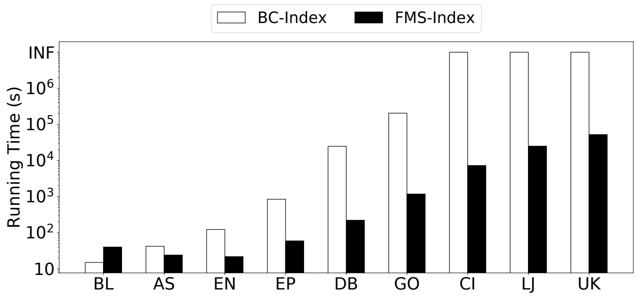


Fig. 12 Time for index construction

construction of FMS–Index, as we have discussed in the "parallelization" of Section 5.2.

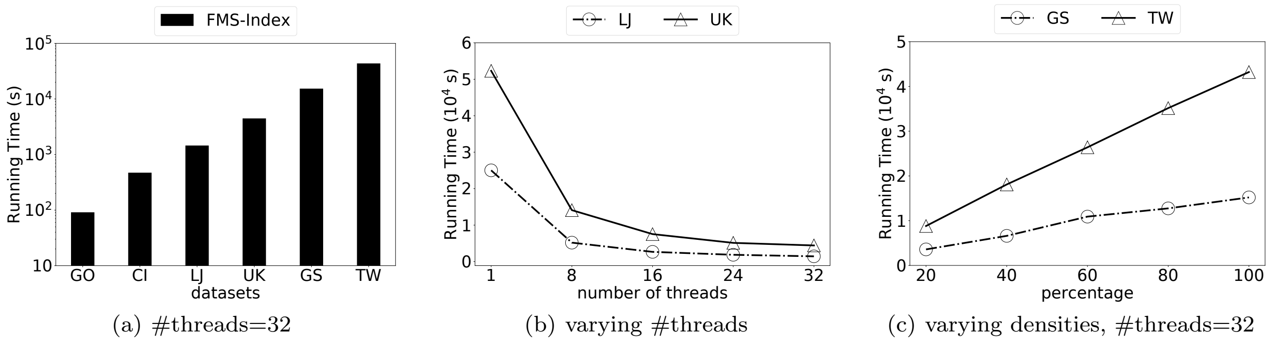
Parallelization and Scalability. We first evaluate constructing the FMS–Index in parallel for all datasets in Figure 13(a). The results for several small datasets are omitted as they are already fast sequentially. We can observe that the performance is significantly improved while adopting parallelization. In particular, for the datasets GS and TW, FMS–Index can not terminate sequentially in time, but it completes within 12 hours while using 32 threads.

In Figure 13(b), we study the scalability of FMS–Index construction by varying the number of threads from 1 to 32 on datasets LJ (a social network) and UK (a web graph). Observe that both curves demonstrate a reasonable drop with the increasing of parallelism, which is sharper at the start from 1 to 8 threads, and eventually becomes smoother due to the cost of thread scheduling.

We then study the scalability by varying the densities on the two largest datasets, i.e., GS and TW, as shown in Figure 13(c). To do so, we randomly sample 20% to 100% edges from the original graph while preserving the number of nodes, and then we construct the subgraphs using the sampled edges. According to Figure 13(c), the performance scales almost linearly with the densities, which conforms with the time complexity analysis of FMS–Index construction (Section 5.2). This also indicates the good scalability of StructSim with the growth of the graph.

Performance of Query Processing. Figure 14 compares the single-pair query performance between NED and StructSim. We exclude RoleSim and IcebergRoleSim here as they are mainly for the all-pair computation. StructSim-BC^b is also omitted as it performs the same as StructSim-BC^f after index construction. For a fair comparison, we set k for StructSim the same as NED. In this test, we randomly select 200 node pairs from each dataset and report the average query time. INF in Figure 14 denotes the case that cannot terminate within one day, and we omit the bars of the out-of-memory cases.

From Figure 14, we can observe that StructSim-Max is faster than NED on all the five datasets (i.e., AS, EN, DB, GO and CI) that NED can complete the computation. Specifically, on datasets AS, EN, and GO, StructSim-Max is significantly faster than NED by 20 times of magnitude. This is because NED involves duplicated nodes in the k-adjacent trees, and it often ends up with larger searching space, while our StructSim-Max adopts the k-hop neighborhood subgraph that avoids the exponential growth of NED's k-adjacent trees. Note that NED cannot terminate within one day for dataset EP, and it fails due to out-of-memory for BL dataset that contains only 10,000 nodes. The culprit is the unbounded size of NED's k-adjacent trees. Both NED and StructSim-Max fail to query on LJ, UK, GS, and TW datasets due to the costly maximum matching, while StructSim-BC succeeds in all cases. Observe that StructSim-BC (without index) is



(a) #threads=32 (b) varying #threads (c) varying densities, #threads=32

Fig. 13 Parallelization and scalability

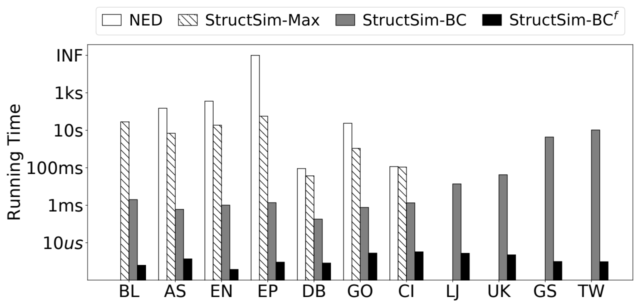


Fig. 14 Query time on different datasets

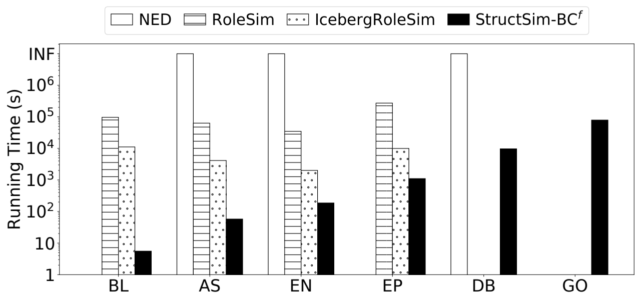


Fig. 15 Time cost for all-pair similarity computation

significantly faster than NED and StructSim-Max with up to five orders of magnitude, which shows the efficiency of BinCount matching (Section 5.1). Specifically, with a precomputed index, the query time of StructSim-BC^f (or StructSim-BC^b) is further improved to less than 10 μs in all cases.

Performance of All-Pairs Computation. Figure 15 reports the all-pairs computation time for NED, RoleSim, IcebergRoleSim and StructSim-BC^f. StructSim-BC^b is omitted here as it performs the same as StructSim-BC^f after index construction. INF in Figure 15 denotes the case that cannot terminate within three days, and we omit the bars of the out-of-memory cases. Note that we do not consider datasets larger than GO in this test because the all-pair computation time is too lengthy.

According to Figure 15, NED fails in the computation on all datasets due to either timeout (i.e., AS, EN and DB) or out-of-memory error (i.e., BL, EP and GO). This is as expected because the size of NED's k-adjacent trees is unbounded, and it needs cubic time complexity to compute a single pair of nodes. RoleSim and IcebergRoleSim perform better than NED, but they still cannot get results on DB and GO datasets due to an out-of-memory error. The $O(n^2)$ space is clearly the culprit. For StructSim-BC^f, it needs only $O(kn \log D)$ space to maintain the index and can be built on all datasets. For efficiency, StructSim-BC^f removes a d^2 term in time complexity when compared with RoleSim. Thus, for the datasets with high average degree, i.e., BL and AS, StructSim-BC^f is faster than RoleSim and IcebergRoleSim with up to 4 orders. As for the datasets EN and EP, StructSim-BC^f still performs at least 1 order faster than IcebergRoleSim. Notably, StructSim-BC^f can finish the all-pair computation for dataset BL within 5 seconds, while it takes RoleSim and IcebergRoleSim several hours even one day to finish the job.

7 Related Work

We review the related work from the following three categories: structural node similarity algorithms, role similarity algorithms, and structural node embeddings.

Structural Node Similarity Algorithms. SimRank [23] is a representative structural node similarity measure with the philosophy "*two nodes are similar if they are related to similar neighbours*". Semantically, two nodes' SimRank value is equivalent to the probability that the random walks from two nodes meet [35,47]. However, SimRank has problems when calculating role similarity. Zheng et al. [59] proved that nodes' SimRank value is negatively correlated with the distance between them, and two disconnected nodes always get SimRank value 0. Zhao et al. [58] found the "Zero-SimRank" issue, and Yu et al. [53] further proved that this issue always happens for the nodes with no symmetric in-link paths between them. However, two nodes that are far away or even disconnected can have similar roles and hence should get high role similarity. Besides, SimRank encounters the "Connectivity Trait" problem [17,56] that the SimRank value between two nodes a and b must be smaller than $\frac{1}{\beta}$, where β is their common in-neighbors. As a result, the SimRank algorithm may give a smaller similarity value for two nodes with more common neighbors, which is obviously anti-intuitive. In [24], the authors showed in the experiments that SimRank performs poorly in computing role similarity. Existing SimRank variants [4,28,31,33,50,52,54,55] either encounter the "Zero-SimRank" issue or the "Connectivity Trait" problem, and thus cannot be applied to calculate role similarity as well. Other algorithms including vertex similarity in [29] and nSimGram in [12] are also not suitable to compute role similarity. The vertex similarity fails to satisfy the "automorphism confirmation" and thus is not a role similarity metric, while the nSimGram is a q-gram-based measure that is designed specifically for node-labeled networks and not appropriate to measure role similarity on the unlabeled graphs.

Role Similarity Algorithms. Jin et al. [24] first formally defined the properties that an admissible role similarity metric should satisfy, among which *automorphism confirmation* is the most important one. RoleSim [24] was proposed as the first admissible role similarity metric, which computes all-pairs of node similarity iteratively and requires a costly maximum matching during computation. IcebergRoleSim [25] was further proposed to speed up the computation of RoleSim by pruning the node pairs whose similarity values are guaranteed to be smaller than a given threshold. NED [60] was originally proposed as a distance metric of nodes. While being normalized to a similarity metric, we prove that it is also a role similarity metric in this paper. Unlike the aforementioned structural node similarity algorithms, RoleSim, IcebergRoleSim, and NED free from the "Zero-SimRank" and "Connectivity Trait" problems, and all of them have the good merit of indicating automorphism (or isomorphism). However, RoleSim, IcebergRoleSim, and NED have severe performance bottlenecks and cannot be scalable to handle large real-world graphs.

Structural Node Embeddings. Node embedding [3, 6,8,9,10,38,40,48] aims at learning a vector representation of each node in the graph to reflect its information (e.g., structural information), which can then be used to compute node similarity. Typical node embedding algorithms, i.e., DeepWalk [39], LINE [46] and node2vec [19] are based on random walks and have the intuition that two nodes have similar embeddings if they are highly possible to co-occur on short random walks over graphs [20]. There are also some other attempts. For example, RolX [21] was proposed to discover the roles of nodes in the networks. struc2vec [41] learned the embeddings of the nodes by performing random walks on a generated multilayer graph. SNS [37] was proposed to combine the information of neighbors and graphlet statistics to compute node embeddings. However, struc2vec and SNS cannot guarantee the automorphism confirmation. GraphWave [15] utilized the spectral wavelets to learn the node embeddings and proved a theoretical bound for the difference between the embeddings of equivalent nodes. Note that our work is orthogonal to node embeddings. On the one hand, embedding approaches can use the role similarity metrics as a learning feature. On the other hand, we can adopt the node embeddings to compute the initial similarity.

8 Conclusion

In this paper, we present a new framework, namely StructSim, to compute nodes' role similarity. Different from the SimRank-based iterative framework, StructSim follows the hierarchical scheme, which adopts the k-neighborhood subgraph to reflect the structural information. Moreover, StructSim admits a precomputed index to better support both the ad-hoc queries and all-pair query, and we further propose an FM-sketch-based technique to build the index efficiently. The extensive experimental results on the real-world graphs demonstrate that StructSim achieves a more effective and efficient role similarity computation and is the only one that can compute role similarity on graphs with billions of edges.

In the future, we plan to do the following two works. First, considering that end-users are more interested in the top-k answers among a huge answer space, it is worth proposing efficient techniques to process top-k queries. Second, attributed graphs become prevalent in various domains (i.e., social networks, web search, and social media) with the proliferation of the world wide web, and thus we plan to investigate calculating role similarity on heterogeneous graphs.

Acknowledgements. Xuemin Lin is supported by NSFC61232006, 2018YFB1003504, ARC DP200101338, ARC DP180103096 and ARC DP170101628. Lu Qin is supported by ARC FT200100787.

References

1. BlogCatalog. `https://github.com/quark0/TAE/tree/master/data/BlogCatalog-dataset`.
2. Optimization and approximation in deterministic sequencing and scheduling: a survey. volume 5 of *Annals of Discrete Mathematics*, pages 287 – 326. 1979.
3. A. Ahmed, N. Shervashidze, S. M. Narayanamurthy, V. Josifovski, and A. J. Smola. Distributed large-scale natural graph factorization. In *Proceedings of the 22nd international conference on World Wide Web*, pages 37–48, 2013.
4. I. Antonellis, H. Garcia-Molina, and C. Chang. Simrank++: query rewriting through link analysis of the click graph. *Proceedings of the VLDB Endowment*, 1(1):408–421, 2008.
5. D. Avis. A survey of heuristics for the weighted matching problem. *Networks*, 13(4):475–493, 1983.
6. M. Belkin and P. Niyogi. Laplacian eigenmaps and spectral techniques for embedding and clustering. In *Advances in Neural Information Processing Systems, NIPS*, pages 585–591, 2001.
7. P. Boldi and S. Vigna. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*, pages 595–602, 2004.
8. S. Cao, W. Lu, and Q. Xu. Grarep: Learning graph representations with global structural information. In *Proceedings of the 24th ACM International Conference on Information and Knowledge Management*, pages 891–900, 2015.
9. S. Cao, W. Lu, and Q. Xu. Deep neural networks for learning graph representations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 1145–1152, 2016.
10. B. P. Chamberlain, J. R. Clough, and M. P. Deisenroth. Neural embeddings of graphs in hyperbolic space. *CoRR*, abs/1705.10359, 2017.
11. X. Chen, L. Lai, L. Qin, and X. Lin. Structsim: Querying structural node similarity at billion scale. In *36th IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1950–1953, 2020.
12. A. Conte, G. Ferraro, R. Grossi, A. Marino, K. Sadakane, and T. Uno. Node similarity with q-grams for real-world labeled networks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 1282–1291, 2018.
13. D. Davis, Ö. N. Yaveroğlu, N. Malod-Dognin, A. Stojmirovic, and N. Pržulj. Topology-function conservation in protein–protein interaction networks. *Bioinformatics*, 31(10):1632–1639, 2015.
14. C. Distinguishability. A theoretical analysis of normalized discounted cumulative gain (ndcg) ranking measures.
15. C. Donnat, M. Zitnik, D. Hallac, and J. Leskovec. Learning structural node embeddings via diffusion wavelets. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1320–1329, 2018.
16. P. Flajolet and G. N. Martin. Probabilistic counting algorithms for data base applications. *Journal of computer and system sciences*, 31(2):182–209, 1985.
17. D. Fogaras and B. Rácz. Scaling link-based similarity search. In *Proceedings of the 14th international conference on World Wide Web*, pages 641–650, 2005.
18. Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for simrank. In *29th IEEE International Conference on Data Engineering*, pages 589–600, 2013.
19. A. Grover and J. Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 855–864, 2016.
20. W. L. Hamilton, R. Ying, and J. Leskovec. Representation learning on graphs: Methods and applications. *IEEE Data Eng. Bull.*, 40(3):52–74, 2017.
21. K. Henderson, B. Gallagher, T. Eliassi-Rad, H. Tong, S. Basu, L. Akoglu, D. Koutra, C. Faloutsos, and L. Li. Rolx: structural role extraction & mining in large graphs. In *The 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1231–1239, 2012.
22. K. Henderson, B. Gallagher, L. Li, L. Akoglu, T. Eliassi-Rad, H. Tong, and C. Faloutsos. It's who you know: graph mining using recursive structural features. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 663–671, 2011.
23. G. Jeh and J. Widom. Simrank: a measure of structural-context similarity. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 538–543, 2002.
24. R. Jin, V. E. Lee, and H. Hong. Axiomatic ranking of network role similarity. In *Proceedings of the 17th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 922–930, 2011.
25. R. Jin, V. E. Lee, and L. Li. Scalable and axiomatic ranking of network role similarity. *ACM Trans. Knowl. Discov. Data*, 8(1):3:1–3:37, 2014.
26. J. M. Kleinberg. Authoritative sources in a hyperlinked environment. *J. ACM*, 46(5):604–632, 1999.
27. H. W. Kuhn. The hungarian method for the assignment problem. In *50 Years of Integer Programming 1958-2008*, pages 29–47. 2010.
28. M. Kusumoto, T. Maehara, and K. Kawarabayashi. Scalable similarity search for simrank. In *Proceedings of the 2014 International Conference on Management of Data*, pages 325–336, 2014.
29. E. A. Leicht, P. Holme, and M. E. Newman. Vertex similarity in networks. *Physical Review E*, 73(2):026120, 2006.
30. J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.
31. C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of simrank for static and dynamic information networks. In *Proceedings of the 13th International Conference on Extending Database Technology*, pages 465–476, 2010.
32. X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *Proceedings of the 23rd International Conference on Data Engineering*, pages 86–95, 2007.
33. Z. Lin, M. R. Lyu, and I. King. Matchsim: a novel neighbor-based similarity measure with maximum neighborhood matching. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 1613–1616, 2009.
34. D. Liu, J. Huang, and C. Lin. Recommendation with social roles. *IEEE Access*, 6:36420–36427, 2018.
35. Y. Liu, B. Zheng, X. He, Z. Wei, X. Xiao, K. Zheng, and J. Lu. Probesim: scalable single-source and top-k

simrank computations on dynamic graphs. *Proceedings of the VLDB Endowment*, 11(1):14–26, 2017.
36. F. Lorrain and H. C. White. Structural equivalence of individuals in social networks. *J. mathematical sociology*, 1(1):49–80, 1971.
37. T. Lyu, Y. Zhang, and Y. Zhang. Enhancing the network embedding quality with structural similarity. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, pages 147–156, 2017.
38. M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu. Asymmetric transitivity preserving graph embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1105–1114, 2016.
39. B. Perozzi, R. Al-Rfou, and S. Skiena. Deepwalk: online learning of social representations. In *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 701–710, 2014.
40. B. Perozzi, V. Kulkarni, and S. Skiena. Walklets: Multiscale graph embeddings for interpretable network classification. *CoRR*, abs/1605.02115, 2016.
41. L. F. Ribeiro, P. H. Saverese, and D. R. Figueiredo. struc2vec: Learning node representations from structural identity. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 385–394, 2017.
42. A. Rosenberg and J. Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL)*, pages 410–420, 2007.
43. R. A. Rossi, B. Gallagher, J. Neville, and K. Henderson. Modeling dynamic behavior in large evolving graphs. In *Sixth ACM International Conference on Web Search and Data Mining*, pages 667–676, 2013.
44. M. A. Serrano and M. Boguná. Topology of the world trade web. *Physical Review E*, 68(1):015101, 2003.
45. A. Strehl and J. Ghosh. Cluster ensembles—a knowledge reuse framework for combining multiple partitions. *Journal of machine learning research*, 3(Dec):583–617, 2002.
46. J. Tang, M. Qu, M. Wang, M. Zhang, J. Yan, and Q. Mei. LINE: large-scale information network embedding. In *Proceedings of the 24th International Conference on World Wide Web*, pages 1067–1077, 2015.
47. B. Tian and X. Xiao. SLING: A near-optimal index structure for simrank. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1859–1874, 2016.
48. D. Wang, P. Cui, and W. Zhu. Structural deep network embedding. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1225–1234, 2016.
49. X. Wang, L. Tang, H. Gao, and H. Liu. Discovering overlapping groups in social media. In *2010 IEEE international conference on data mining*, pages 569–578. IEEE, 2010.
50. Y. Wang, X. Lian, and L. Chen. Efficient simrank tracking in dynamic graphs. In *2018 IEEE 34th International Conference on Data Engineering*, pages 545–556, 2018.
51. S. Wasserman and K. Faust. *Social network analysis: Methods and applications*, volume 8. Cambridge university press, 1994.
52. W. Yu, X. Lin, and W. Zhang. Towards efficient simrank computation on large networks. In *29th IEEE International Conference on Data Engineering*, pages 601–612, 2013.
53. W. Yu, X. Lin, W. Zhang, L. Chang, and J. Pei. More is simpler: Effectively and efficiently assessing node-pair similarities based on hyperlinks. *Proceedings of the VLDB Endowment*, 7(1):13–24, 2013.
54. W. Yu, X. Lin, W. Zhang, J. Pei, and J. A. McCann. Simrank*: effective and scalable pairwise similarity search based on graph topology. *The VLDB Journal*, 28(3):401–426, 2019.
55. W. Yu and J. A. McCann. Efficient partial-pairs simrank search for large networks. *Proceedings of the VLDB Endowment*, 8(5):569–580, 2015.
56. W. Yu and J. A. McCann. High quality graph-based similarity search. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 83–92, 2015.
57. K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information processing letters*, 42(3):133–139, 1992.
58. P. Zhao, J. Han, and Y. Sun. P-rank: a comprehensive structural similarity measure over information networks. In *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, pages 553–562, 2009.
59. W. Zheng, L. Zou, Y. Feng, L. Chen, and D. Zhao. Efficient simrank-based similarity join over large graphs. *Proceedings of the VLDB Endowment*, 6(7):493–504, 2013.
60. H. Zhu, X. Meng, and G. Kollios. NED: an inter-graph node metric based on edit distance. *Proceedings of the VLDB Endowment*, 10(6):697–708, 2017.