RoleSim+: A Fast Algorithm for RoleSim Similarity Search

Weiren Yu[†], Sima Iranmanesh[†], Xuming Hong[♯], Jianxun Xu[♯]

[†]Warwick University, Coventry, UK [♯]Nanjing University of Sci. & Tech., Jiangsu, China
{weiren.yu, sima.iranmanesh}@warwick.ac.uk {hongxuming, xjxcst}@njust.edu.cn

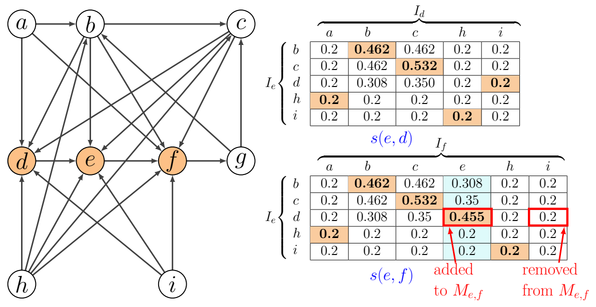
ABSTRACT

The conundrum of quantifying pairwise similarity based on network topology arises in many graph mining applications, *e.g.,* community detection. RoleSim is an arresting similarity measure that recursively defines similarity between entities as the average similarity of the maximum weight matching between their neighbours. Nonetheless, the existing algorithm [4] to compute RoleSim similarity is expensive due to many duplicate computations over different pairs of nodes. In this paper, we propose a novel efficient algorithm, RoleSim+, which accelerates the computation of RoleSim without loss of accuracy. To avoid duplicate computations, unlike RoleSim that computes the maximum weight matching for each pair of nodes independently from scratch, RoleSim+ employs a novel method that resorts to a Steiner tree to find an optimised topological sorting, aiming at maximising the reuse of the previously computed maximum matching information, progressively. Our experimental evaluations on various real datasets validate the superiority of RoleSim+ at a decent speedup without any compromise of exactness.

1 INTRODUCTION

An overarching task in graph mining is to evaluate pairwise similarity between nodes using the link structure of a network. This technique, also known as *link analysis*, has been very popular in a myriad of real applications, *e.g.,* collaborative filtering, community detection, and query rewriting. In contrast to the content-based similarity that focuses on the text attributes of the objects, link-based similarity hinges on the link structure of the graph. Amid piles of existing link-based models [2, 9, 12, 13, 21], RoleSim, conceptualised by Jin *et al.* [3], has surfaced as an attractive one on account of its concise and intuitive philosophy that "two objects are assessed as similar if their in-neighbors are automorphically similar". Similar to SimRank measure [2], RoleSim can recursively capture multi-hop neighbouring information of two nodes. However, RoleSim, as its name indicates, is superior to SimRank for identifying nodes with similar roles. This is because, unlike SimRank similarity $s(a,b)$ that is defined as the average value of all the pairwise similarities of node a's and b's in-neighbors, RoleSim similarity $s(a,b)$ considers the average value of only the maximum weight matching of a's and b's in-neighbors. As a result, despite no connected paths between nodes, RoleSim can assess the nodes as similar if their roles are similar. Hence, RoleSim has many applications in, *e.g.,* co-authorship analysis [14], and social network de-anonymization [10].

However, the computational time of RoleSim is still undesirable due to duplicate computations, as shown in Example 1.1.



	a	b	c	h	i
b	0.2	**0.462**	0.462	0.2	0.2
c	0.2	0.462	**0.532**	0.2	0.2
d	0.2	0.308	0.350	0.2	**0.2**
h	**0.2**	0.2	0.2	0.2	0.2
i	0.2	0.2	0.2	**0.2**	0.2

	a	b	c	e	h	i
b	0.2	**0.462**	0.462	0.308	0.2	0.2
c	0.2	0.462	**0.532**	0.35	0.2	0.2
d	0.2	0.308	0.35	**0.455**	0.2	0.2
h	**0.2**	0.2	0.2	0.2	0.2	0.2
i	0.2	0.2	0.2	0.2	**0.2**	0.2

Figure 1: Duplicate Computations in $s(e,d)$ **&** $s(e,f)$

Example 1.1. Consider a web graph G in Figure 1, where each edge is a hyperlink. Given decay factor $\beta = 0.8$, we want to evaluate RoleSim similarities $s(e,d)$ and $s(e,f)$ in G.

For any node x in G, let I_x be the in-neighbor set of node x in G, and $|I_x|$ the cardinality of I_x. For in-neighbor sets I_x and I_y, let $M(I_x, I_y)$ be the maximum weight matching from the pairwise RoleSim similarities over x's and y's in-neighbor grid $I_x \times I_y$.

The existing RoleSim algorithm [4] computes $s(e,d)$ and $s(e,f)$ in three steps: First, it finds the maximum weight matching $M(I_e, I_d)$ (*resp.* $M(I_e, I_f)$) from the similarities of the in-neighbor grid $I_e \times I_d$ (*resp.* $I_e \times I_f$), as colored in the five orange cells.

Then, we add up scores on each matching $M(I_e, I_d)$ (*resp.* $M(I_e, I_f)$) from scratch, yielding the total weight $\omega(I_e, I_d)$ (*resp.* $\omega(I_e, I_f)$):

$$\begin{aligned}\omega(I_e, I_d) &= s(b,b) + s(c,c) + s(h,a) + s(i,h) + s(d,i)\\ &= \underline{0.462 + 0.532 + 0.2 + 0.2} + 0.2 = 1.594 \qquad (1a)\end{aligned}$$

$$\begin{aligned}\omega(I_e, I_f) &= s(b,b) + s(c,c) + s(h,a) + s(i,h) + s(d,e)\\ &= \underline{0.462 + 0.532 + 0.2 + 0.2} + 0.455 = 1.849 \qquad (1b)\end{aligned}$$

By RoleSim definition [4], $s(e,d)$ and $s(e,f)$ are computed:

$$s(e,d) = \beta \times \frac{\omega(I_e,I_d)}{\max\{|I_e|,|I_d|\}} + (1-\beta) = 0.8 \times \frac{1.594}{5} + 0.2 = 0.455$$

$$s(e,f) = \beta \times \frac{\omega(I_e,I_f)}{\max\{|I_e|,|I_f|\}} + (1-\beta) = 0.8 \times \frac{1.849}{6} + 0.2 = 0.447$$

During the above process, we perceive that, in Eqs.(1a) and (1b), there are duplicate computations for $\omega(I_e, I_d)$ and $\omega(I_e, I_f)$ (underlined parts). If the result of the weight $\omega(I_e, I_d)$ in Eq.(1a) is cached and reused later for $\omega(I_e, I_f)$ computation in Eq.(1b):

$$\omega(I_e, I_f) = \omega(I_e, I_d) - s(d,i) + s(d,e) = 1.594 - 0.2 + 0.455 = 1.849,$$

then a vast amount of time can be saved for $s(e,f)$ search. □

Example 1.1 implies that, when each pair of RoleSim similarity is computed from scratch, such "shared-nothing" algorithms are inefficient due to many repetitive calculations. However, there is no regularity in the overlap of neighboring nodes on a real graph, calling for efficient sharing strategies for fast RoleSim search.

Contributions. Our main contributions are as follows:

- We first devise an efficient computation sharing strategy, which resorts to a Steiner tree to find an optimised topological sorting, aiming at maximising the reuse of the previously computed maximum matching information progressively. (Section 3.1)
- Based on our sharing strategy, we next propose a fast algorithm, RoleSim+, to compute RoleSim similarities efficiently, unlike the existing algorithm [4] that computes the maximum matching for each pair of nodes independently from scratch. (Section 3.2)
- We conduct experiments on real datasets to validate that our proposed RoleSim+ is consistently faster than the best-known competitor without any loss of accuracy. (Section 4)

Related Work. Recent years have witnessed a growing interest in RoleSim-like similarity search [3, 4, 10, 14–20]. Jin *et al.* [3, 4] devised an Iceberg algorithm, which prunes unpromising pairs to identify nodes with the most interesting connections, thus reducing computational overheads for RoleSim search. Shao *et al.* [10] proposed α-RoleSim++, which discards tiny iterative similarities below a user-specified threshold. However, these approaches are approximate, which compromises a little accuracy for speedups. In comparison, we provide novel accurate and efficient methods that leverage a Steiner tree to find an optimised topological sorting for maximising the reuse of the previously computed maximum weight matching, which enables a decent speed-up without loss of exactness. Our method can also be incorporated to Iceberg [4] and α-RoleSim++ [10] to further speed up their similarity search algorithms.

There has also been a host of work on variations of RoleSim, *e.g.,* MatchSim [7], CentSim [6], RoleSim++ [10], and RoleSim* [14]. MatchSim [7] is a RoleSim-like model that adopts a different initialisation approach, but may not guarantee automorphic equivalence. CentSim [6] is another role-based similarity measure that compares the centrality values of vertices. It exploits the weighted average of PageRank, degree and closeness centrality to define similarity. RoleSim++ [10] encodes the maximum matching of both in- and out-neighbors into similarity values, whose search quality is higher than RoleSim for de-anonymisation of social networks. Recently, RoleSim* [14], an enhanced version of RoleSim, is proposed, which combines the merits of both SimRank and RoleSim. Its similarities can well capture not only the automorphic equivalence of two nodes, but also other neighboring similarities outside the automorphically equivalent sets which are overlooked by RoleSim. However, the main focuses of these studies are on search quality.

2 PRELIMINARIES

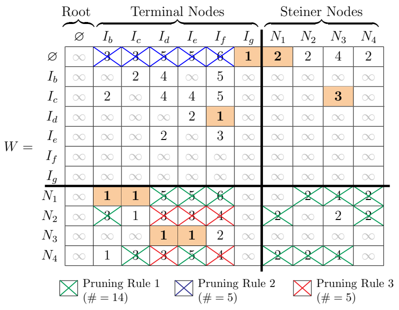
Given a digraph $G = (V, E)$ with a node set V and an edge set E, the RoleSim similarity $s(a, b)$ between nodes a and b is defined as

$$s(a,b) = \frac{\beta \times \omega(I_a, I_b)}{\max\{|I_a|, |I_b|\}} + (1-\beta) \ \text{ with } \ \omega(I_a, I_b) = \sum_{(x,y)\in M(I_a,I_b)} s(x,y) \quad (2)$$

where $\beta \in (0, 1)$ is a decay factor, and $M(I_a, I_b)$ is the maximum weight matching of a bipartite graph $B = (I_a \cup I_b, I_a \times I_b, s(*, *))$. Each edge (x, y) in B connecting node $x \in I_a$ and node $y \in I_b$ has an associated weight $s(x, y)$, whose value is the RoleSim similarity between nodes x and y in G. $\omega(I_a, I_b)$ is the sum of the edge weights (*i.e.,* RoleSim scores) over the matching $M(I_a, I_b)$ in B.

To compute $s(a, b)$, Jin *et al.* [3] proposed the following iterative approach: Initially, $s_0(a, b) = 1$ ($\forall a, b$). At $(k + 1)$-th iteration,

$$s_{k+1}(a,b) = \frac{\beta \times \omega_k(I_a, I_b)}{\max\{|I_a|, |I_b|\}} + (1-\beta) \ \text{ with } \ \omega_k(I_a, I_b) = \sum_{(x,y)\in M_k(I_a,I_b)} s_k(x,y) \quad (3)$$



$$W = \begin{array}{c|c|cccccc|cccc}
 & \text{Root} & \multicolumn{6}{c|}{\text{Terminal Nodes}} & \multicolumn{4}{c}{\text{Steiner Nodes}} \\
 & \varnothing & I_b & I_c & I_d & I_e & I_f & I_g & N_1 & N_2 & N_3 & N_4 \\
\hline
\varnothing & \infty & \xcancel{3} & \xcancel{3} & \xcancel{5} & \xcancel{5} & \xcancel{6} & \mathbf{1} & \mathbf{2} & 2 & 4 & 2 \\
I_b & \infty & \infty & 2 & 4 & \infty & 5 & \infty & \infty & \infty & \infty & \infty \\
I_c & \infty & 2 & \infty & 4 & 4 & 5 & \infty & \infty & \infty & \mathbf{3} & \infty \\
I_d & \infty & \infty & \infty & \infty & 2 & \mathbf{1} & \infty & \infty & \infty & \infty & \infty \\
I_e & \infty & \infty & \infty & 2 & \infty & 3 & \infty & \infty & \infty & \infty & \infty \\
I_f & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
I_g & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty \\
\hline
N_1 & \infty & \mathbf{1} & \mathbf{1} & \xcancel{5} & \xcancel{5} & \xcancel{6} & \infty & \infty & \xcancel{2} & \xcancel{4} & \xcancel{2} \\
N_2 & \infty & \xcancel{3} & 1 & \xcancel{3} & \xcancel{3} & \xcancel{4} & \infty & \xcancel{2} & \infty & 2 & \xcancel{2} \\
N_3 & \infty & \infty & \infty & \mathbf{1} & \mathbf{1} & 2 & \infty & \infty & \infty & \infty & \infty \\
N_4 & \infty & 1 & \xcancel{3} & \xcancel{3} & \xcancel{5} & \xcancel{4} & \infty & \xcancel{2} & \xcancel{2} & \xcancel{4} & \infty
\end{array}$$

⊠ Pruning Rule 1 (# = 14) ⊠ Pruning Rule 2 (# = 5) ⊠ Pruning Rule 3 (# = 5)

Figure 2: Transitional Cost Matrix W

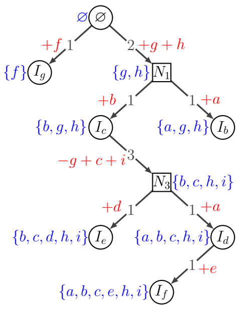


Figure 3: Steiner Tree $\mathscr{T}$

where $s_{k+1}(a, b)$ is the $(k+1)$-th iterative similarity, and $M_k(I_a, I_b)$ is the maximum weight matching of $B_k = (I_a \cup I_b, I_a \times I_b, s_k(*, *))$. It was shown in [4] that $s_k(a, b)$ converges to $s(a, b)$ as $k \to \infty$. The time complexity of the iterative method is $O(K|V|^2 d^3)$ on G with $|V|$ nodes for K iterations, where d is the average degree.

3 PROPOSED SOLUTION

3.1 Computation Sharing Strategy

We first find an optimised topological sorting that can efficiently reuse the previously computed maximum matching information.

Our first step is to construct a cost transition graph $\mathscr{G} = (\mathscr{V}, \mathscr{E}, W)$ from G. The vertex set of $\mathscr{G}$ is $\mathscr{V} = \{\varnothing\} \cup \mathcal{I} \cup \mathcal{I}^{(2)}$, where $\mathcal{I}$ is a collection of non-empty in-neighbor sets I_x of each node x in G, and $\mathcal{I}^{(2)}$ is a collection of the intersections of any 2 sets in $\mathcal{I}$. For any two sets X and Y in $\mathscr{V}$, if $|X| \leq |Y|$, then there is an edge $X \to Y$ in $\mathscr{E}$ associated with the weight $W(X, Y)$ defined as follows:

$$W(X,Y) = \begin{cases} |X| + |Y| - 2|X \cap Y|, & \text{if } X \not\subseteq Y \text{ and } |X| \leq \min\{2|X \cap Y|, |Y|\} \\ |Y| - |X|, & \text{if } X \subseteq Y \\ \infty, & \text{otherwise} \end{cases} \quad (4)$$

Intuitively, $W(X, Y)$ is the transitional cost from X to Y, *i.e.,* the number of operations required to generate Y from X, where each operation refers to adding/removing an element to/from X. For example, if $X = \{a, b, c, h, i\}$ and $Y = \{b, c, d, h, i\}$, then $W(X, Y) = |X| + |Y| - 2|X \cap Y| = 5 + 5 - 2 \times 4 = 2$, meaning that 2 operations are needed to generate Y from X, *i.e.,* $Y = X - \{a\} \cup \{i\}$. It is worth noting that $W(X, Y)$ actually reflects how many elements of X can be reused to generate Y. The smaller the value of $W(X, Y)$, the larger the overlap between X and Y. In an extreme case when $W(X, Y) = 0$, X can be fully reused to generate Y (due to $X = Y$). $W(X, Y) = \infty$ means that edge $X \to Y$ is not added to $\mathscr{E}$ since the $W(X, Y)$ operations required to get Y from X via Eq.(4) is more costly than the $|Y|$ operations to generate Y from scratch ($\varnothing$).

Our second step is to find a Steiner tree $\mathscr{T}$ from $\mathscr{G}$, which will produce an optimised topological sorting for efficient computation sharing to generate all sets in $\mathcal{I}$ with minimum transitional costs. Precisely, given a weighted graph $\mathscr{G}(\mathscr{V}, \mathscr{E}, W)$ with a root $\{\varnothing\}$ and a subset $\mathcal{I}$ of $\mathscr{V}$ (where each set in $\mathcal{I}$ is called *a terminal*), our goal is to find a minimum cost arborescence $\mathscr{T}$ that connects the root $\{\varnothing\}$ to each terminal in $\mathcal{I}$. The remaining vertices in $\mathscr{V} - \mathcal{I} - \{\varnothing\}$ that are used to construct the Steiner tree $\mathscr{T}$ are called *Steiner vertices.*

Intuitively, each path in $\mathscr{T}$ from $\{\varnothing\}$ to any terminal in $\mathcal{I}$ implies a topological sorting for efficient computation sharing. Each edge $X \xrightarrow{w} Y$ in $\mathscr{T}$ means that Y can be generated from X through only w operations. This indicates that the generation of Y does not have to start from scratch by adding $|Y|$ elements to $\varnothing$ since part of the elements of X can be reused to generate Y with only w operations being required, where $w = W(X, Y)$ ($< |Y|$) is defined by Eq.(4).

Example 3.1 (Computation sharing using Steiner Tree). Recall G in Figure 1. Our computational sharing method follows 3 steps:

First, we build a transitional graph $\mathscr{G}(\mathscr{V}, \mathscr{E}, W)$ with $\mathscr{V} = \{\varnothing\} \cup \mathcal{I} \cup \mathcal{I}^{(2)}$. $\mathcal{I} = \{I_b, I_c, I_d, I_e, I_f, I_g\}$, where $I_b = \{a, g, h\}$, $I_c = \{b, g, h\}$, $I_d = \{a, b, c, h, i\}$, $I_e = \{b, c, d, h, i\}$, $I_f = \{a, b, c, e, h, i\}$, $I_g = \{f\}$; and $\mathcal{I}^{(2)} = \{N_1, N_2, N_3, N_4\}$, where $N_1 = \{g, h\}$, $N_2 = \{b, h\}$, $N_3 = \{b, c, h, i\}$, $N_4 = \{a, h\}$. For instance, N_1 is the intersection of I_b and I_c.

Using Eq.(4), we can obtain the transitional cost matrix W of $\mathscr{G}$, as shown in Figure 2. Each cell in W, whose value is not '∞', corresponds to a directed edge in $\mathscr{E}$. For instance, '2' indexed at row 'I_b' and column 'I_c' corresponds to a directed edge $I_b \longrightarrow I_c$ in $\mathscr{E}$ with a weight of 2.

Next, we find a Steiner tree $\mathscr{T}$ (with the root $\{\varnothing\}$ and terminals $\mathcal{I}$) that spans $\mathscr{G}$, as depicted in Figure 3. Each edge in $\mathscr{T}$ corresponds to a weight colored in an orange cell of W. The sum of all edge weights in $\mathscr{T}$ ($2 + 1 + 1 + 1 + 3 + 1 + 1 + 1 = 11$) is the minimum cost.

Particularly, each path in $\mathscr{T}$ from the root $\{\varnothing\}$ to any terminal in $\mathcal{I}$ implies a topological sorting describing how shared computing works. For example, paths $\varnothing \xrightarrow{2} N_1 \xrightarrow{1} I_b$ and $\varnothing \xrightarrow{2} N_1 \xrightarrow{1} I_c$ in $\mathscr{T}$ indicate that N_1 is the common part that is created from scratch only once, and can be reused twice for getting both terminals I_b and I_c. Thus, the total cost of our method for generating I_b and I_c is 4 ($= 2 + 1 + 1$). □

Shared Maximum Weight Matching. The topological sorting in $\mathscr{T}$ implies an efficient order of incrementally finding a series of maximum weight matchings, as shown in Theorem 3.2:

THEOREM 3.2. *Each edge $X \xrightarrow{w} Y$ in the Steiner tree $\mathscr{T}$ indicates that, for any subset Z of nodes in G, the maximum weight matching $M(Z, Y)$ and its maximum weight $\omega(Z, Y) := \sum_{(z,y)\in M(Z,Y)} s(z, y)$ in bipartite graph $B_1 = (Z \cup Y, Z \times Y, s(*, *))$ can be incrementally obtained from the maximum weight matching $M(Z, X)$ and its maximum weight $\omega(Z, X)$ in $B_2 = (Z \cup X, Z \times X, s(*, *))$, using only $O(wl^2)$ time, where $l = \max\{|X|, |Y|, |Z|\}$, w ($< l$) is edge weight.*

PROOF. For edge $X \xrightarrow{w} Y$ in $\mathscr{T}$, and any subset Z of nodes in G, let $s(Z, X) = \{s(z, x) \mid \forall z \in Z \text{ and } \forall x \in X\}$ be the RoleSim similarity grid with node-pairs over $Z \times X$, and $s(Z, Y)$ be the similarity grid over $Z \times Y$. Since $X \xrightarrow{w} Y$ implies that Y can be generated from X with only w operations, $s(Z, Y)$ can be thought of as the update of w columns to $s(Z, X)$. Thus, given the maximum weight matching $M(Z, X)$ over the grid $s(Z, X)$, the incremental assignment algorithm [8, 11] can be used to determine the maximum weight matching $M(Z, Y)$ over the updated grid $s(Z, Y)$, entailing only $O(wl^2)$ time with $w = W(X, Y)$ in Eq.(4) and $l = \max\{|X|, |Y|, |Z|\}$. □

Example 3.3. Recall G in Figure 1 and its Steiner tree $\mathscr{T}$ in Figure 3. To efficiently compute RoleSim similarities $s(e, d)$ and $s(e, f)$ in G, we notice that edge $I_d \longrightarrow I_f$ exists in $\mathscr{T}$, implying that $I_d \cup \{e\} = I_f$. Thus, the maximum matching $M(I_e, I_f)$ and its weight $\omega(I_e, I_f)$ can be incrementally computed from $M(I_e, I_d)$ and $\omega(I_e, I_d)$ using only $O(l^2)$ time, where $l = \max\{|I_d|, |I_e|, |I_f|\}$, through the incremental assignment algorithm [8, 11], unlike the existing method in Example 1.1 that requires $O(l^3)$ to find $M(I_e, I_f)$ and $M(I_e, I_d)$ from scratch. □

Sparsification of $\mathscr{G}$. The transitional cost matrix W generated by Eq.(4) is rather dense. To sparsify $\mathscr{G}$, we propose 3 pruning rules to discard unpromising edges from $\mathscr{G}$, prior to getting $\mathscr{T}$. These prunings are lossless since removing such edges from $\mathscr{G}$ will not affect the result of finding $\mathscr{T}$ from $\mathscr{G}$ while reducing the total weight of the retrieving result towards $\mathscr{T}$ with the minimum weight.

Algorithm 1: TraverseT($\mathscr{T}, X, \beta, s_k(I_v, *)$)

```
Input  : $\mathscr{T}$: Steiner tree, $X$: vertex in $\mathscr{T}$, $\beta$: decay factor, $v$: query
         $s_k(I_v, *)$: similarities $\{s_k(x, y)\}_{\forall(x,y)\in I_v\times V}$ at iteration $k$
Output : $s_{k+1}(v, *)$: RoleSim similarities w.r.t. $v$ at iteration $k+1$
1  if $X \neq \varnothing$ then
2      build a bipartite graph $B_k := (I_v \cup X, I_v \times X, s_k(*, *))$
3      $Z := \mathscr{T}.parent(X)$
4      if $Z = \varnothing$ then
5          $[\omega_k(I_v, X); M_k(I_v, X)] :=$ MaxWeight($B_k$);
6      else
7          $[\omega_k(I_v, X); M_k(I_v, X)] :=$ dynMaxWeight($B_k, M_k(I_v, Z)$);
8  if $X \in \mathcal{I}$ then
9      set $u :=$ node in $V$ whose in-neighbor set is $X$
10     $s_{k+1}(v, u) := \beta \times \frac{\omega_k(I_v, X)}{\max\{|I_v|, |X|\}} + (1 - \beta)$;
11 foreach child $Y$ of $X$ in $\mathscr{T}$ do
12     TraverseT ($\mathscr{T}, Y, \beta, s_k(I_v, *)$);
13 return $s_{k+1}(v, *)$;
```

Pruning Rule 1. For any two sets X and Y in $\mathscr{V}$, edge $X \xrightarrow{w} Y$ in $\mathscr{G}$ can be pruned if $w \geq |Y|$. □

Pruning Rule 1 indicates that, if the cost of generating Y from X is no less than $|Y|$, we will generate Y from scratch ($\varnothing$), and prune $X \longrightarrow Y$ in $\mathscr{G}$. For example in Figure 2, $N_2 \longrightarrow I_b$ can be pruned in W, as denoted by a green cross, as $|I_b| = 3 \leq W(N_2, I_b)$.

Pruning Rule 2. For any set $Y \in \mathcal{I}$, edge $\varnothing \longrightarrow Y$ in $\mathscr{G}$ can be pruned if there exists $Z \in \mathscr{V}$ s.t. $W(Z, Y) \leq |Y|$. □

Pruning Rule 2 implies that, if $\exists$ a node $Z \in \mathscr{V}$ *s.t.* the cost of generating Y from Z is no more than $|Y|$ (*i.e.,* the cost of generating Y from $\varnothing$), then we will not consider to generate Y from $\varnothing$ and prune $\varnothing \longrightarrow Y$. For instance in Figure 2, $\varnothing \longrightarrow I_b$ can be pruned in W, indicated by a blue cross, since $\exists I_c \in \mathscr{V}$ *s.t.* $W(I_c, I_b) = 2 < 3 = |I_b|$.

Pruning Rule 3. For any set $X \in \mathcal{I}^{(2)}$ and $Y \in \mathscr{V}$, edge $X \xrightarrow{w} Y$ in $\mathscr{G}$ can be pruned if there exists $Z \in \mathcal{I}^{(2)}$ *s.t.* $W(Z, Y) \leq w$. □

Pruning Rule 3 indicates that, for any two sets X and Z in $\mathcal{I}^{(2)}$, if the cost of generating Y from Z is less than w (*i.e.,* the cost of generating Y from X), $X \longrightarrow Y$ will be pruned in $\mathscr{G}$. For example in Figure 2, edge $N_2 \longrightarrow I_d$ is pruned in W, denoted as a red cross, because there exists $N_3 \longrightarrow I_d$ *s.t.* $W(N_3, I_d) = 1 < 3 = W(N_2, I_d)$.

3.2 A Complete Algorithm

Traversing $\mathscr{T}$ to Accelerate RoleSim Computation. We first provide an efficient algorithm to traverse $\mathscr{T}$ based on depth-first search, aiming at speeding up RoleSim computation.

TraverseT starts at the root ($\varnothing$) of $\mathscr{T}$, and explores as far as possible along each branch of $\mathscr{T}$ before backtracking. If a visited node X in $\mathscr{T}$ is not the root ($\varnothing$) (line 1), the weighted bipartite graph B_k over the grid $I_v \times X$ will be built first (line 2), and then the maximum weight matching $M_k(I_v, X)$ and its weight $\omega_k(I_v, X)$ over B_k are computed in two different ways, depending on the parent node Z of X: If Z is the root ($\varnothing$) of $\mathscr{T}$, $M_k(I_v, X)$ is computed from scratch in a conventional way using an $O(d^3)$-time Kuhn-Munkres algorithm (MaxWeight) [1], where d is the average degree of G; otherwise, a dynamic $O(wd^2)$-time (with $w = W(Z, X) \ll d$) maximum weight matching algorithm (dynMaxWeight) [8, 11] is employed to incrementally compute $M_k(I_v, X)$ and $\omega_k(I_v, X)$ from the previously computed matching $M_k(I_v, Z)$ (lines 3–7), thereby achieving a speedup over RoleSim that computes $M_k(I_v, X)$ and $\omega_k(I_v, X)$ from scratch. When X is

Algorithm 2: RoleSim+(G, β, K)

```
    Input  : $G = (V, E)$: graph, $\beta$: decay factor, $K$: # of iterations
    Output : $s_K(*, *)$: RoleSim similarities after $K$ iterations
1   initialise $s_0(*, *) := 1$;
2   $\mathcal{I} := \{I_x \neq \varnothing \mid x \in V\}$, $\mathcal{I}^{(2)} :=$ {intersections of any 2 sets in $\mathcal{I}$};
3   $\mathscr{V} := \{\varnothing\} \cup \mathcal{I} \cup \mathcal{I}^{(2)}$,   $\mathscr{E} := \varnothing$;
4   foreach $(X, Y) \in \mathscr{V} \times \mathscr{V}$ do
5       compute $W(X, Y)$ according to Eq.(4);
6       if $W(X, Y) \neq \infty$ & does not violate Pruning Rules 1–3 then
7           add edge $(X \to Y)$ into $\mathscr{E}$;
8   $\mathscr{T}$ := getSteinerTree ($\mathscr{G}(\mathscr{V}, \mathscr{E}, W)$);
9   for $k := 1, 2, \cdots, K$ do
10      foreach $v \in V$ do
11          $s_k(v, *)$ :=TraverseT ($\mathscr{T}, \varnothing, \beta, s_{k-1}(I_v, *)$);
12  return $s_K(*, *)$;
```

a terminal in $\mathscr{T}$ (*i.e.,* $\exists$ node u in G whose in-neighbor set is X), then we compute $s_{k+1}(v, u)$ from $\omega_k(I_v, X)$ (lines 8–10). After node X is processed, we iterate over its children in $\mathscr{T}$ and call `TraverseT` $(\cdot)$ recursively (lines 11–12).

A Complete Algorithm. Combining `TraverseT`$(\cdot)$ with all techniques in Section 3.1, Algorithm 2 shows our complete scheme.

Computational Complexity. The time of Algorithm 2 consists of two phases: I) In the Preprocessing phase, it requires $O(|\mathcal{I}|d + |\mathcal{I}^{(2)}|d)$ time to get $\mathcal{I}$ and $\mathcal{I}^{(2)}$ (line 2), and $O(|\mathscr{V}|^2)$ time to obtain a weight matrix W (lines 4–7), where $|\mathscr{V}|$ is the number of nodes in $\mathscr{G}$; and d is the average degree of G. Then, it takes $O(|\mathscr{E}| \log |\mathscr{V}|)$ time to get a Steiner tree $\mathscr{T}$ from $\mathscr{G}$. II) In the Iterative Shared Computing phase, for each iteration k and each query v, it entails $O(|\mathscr{T}_\varnothing|d^3 + |\mathscr{T}_V|d^2 w)$ time to compute $s_k(v, *)$ by traversing the Steiner tree $\mathcal{T}$ (line 11), where $|\mathscr{T}_\varnothing|$ is the number of out-neighhors of $\varnothing$ in $\mathscr{T}$, $|\mathscr{T}_V|$ is the number of remaining nodes in $\mathscr{T}$, and w is the total minimum weight of $\mathscr{T}$. Based on our analysis in Algorithm 1, the time for `TraverseT`$(\cdot)$ is $O(\sum_{X \in \mathscr{T}_\varnothing} d^3 + \sum_{X \in \mathscr{T}_V} d^2 W(Z, X)) = O(|\mathscr{T}_\varnothing|d^3 + |\mathscr{T}_V|d^2 w))$ per iteration. Combining I) and II), we get the total time complexity of Algorithm 2, which is bounded by $O(|\mathscr{V}|^2 + |\mathscr{E}| \log |\mathscr{V}| + K|V|(|\mathscr{T}_\varnothing|d^3 + |\mathscr{T}_V|d^2 w))$.

4 EXPERIMENTS

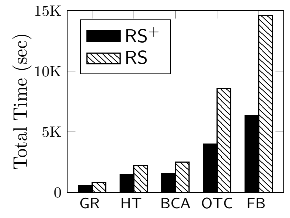
Experimental Settings. We used real datasets from SNAP [5].

Datasets	Abbr	$\lvert V\rvert^2$ Pairs	$\lvert E\rvert$	Description
CA-GrQc	GR	27,478,564	14,496	Arxiv General Relativity Coauthorship
CA-HepTh	HT	97,555,129	25,998	Arxiv HEP Theory Collaboration
Bitcoin-α	BCA	14,311,089	24,186	Bitcoin Alpha web of trust network
Bitcoin-OTC	OTC	34,586,161	35,592	Bitcoin OTC web of trust network
Ego-Facebook	FB	16,313,521	88,234	Social circles from Facebook

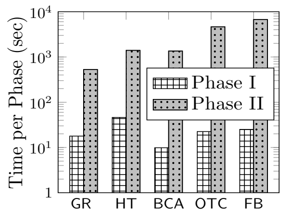
We implemented the following algorithms in Visual C++.NET. i) RS$^+$, our fast method in Algorithm 2. ii) RS [4], the best-known algorithm for iteratively computing RoleSim similarities. All the algorithms are run on Windows 10 with an Intel Core i7-10700 CPU @ 2.9GHz and 16GB RAM. As previously used in [4], we set decay factor $\beta = 0.8$, and number of iterations $K = 5$ by default.

Experimental Results We next present our findings.

(1) Time Efficiency. Fig. 4a shows the high computational efficiency of RS$^+$ on five real datasets. We observe that (i) on each dataset, the total time of RS$^+$ is consistently 1.5x–2.3x faster than that of RS. This highlights the effectiveness of our approach that resorts to a Steiner tree to find an optimised topological sorting for maximising the reuse of the the previously computed maximum matching. In contrast, RS is a "shared-nothing" algorithm that involves many repetitive calculations to evaluate each pair of similarity independently from scratch. (ii) When the density of the graph is growing, the speedup of RS$^+$ relative to RS is more pronounced. For example, on high-density graphs (*e.g.,* FB with $d = 21.84$), RS$^+$ is 2.3x faster than RS, whereas on



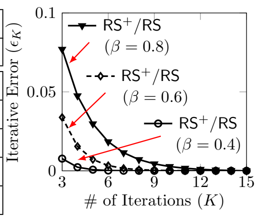
(a) Total Time (RS$^+$ *vs.* RS)



(b) Time per Phase for RS$^+$

Data	# of Pairs Pruned by P1–P3				
	P1	**P2**	**P3**	**Total**	**(%)**
GR	13,182	4,400	29	17,611	41.1
HT	30,229	8,244	113	38,586	50.8
BCA	90,669	3,784	185	94,638	23.5
OTC	193,393	5,882	324	199,599	23.7
FB	193,172	4,040	5	197,217	40.6

(c) Pruning Power of RS$^+$



(d) Accuracy & Exactness

Figure 4: Experimental Results on Real Datasets

low-density graphs (*e.g.,* HT with $d = 2.63$), RS$^+$ is 1.5x faster than RS. This is consistent with our time complexity analysis of RS$^+$ in Algorithm 2, where each time RS$^+$ visits a non-terminal node X during $\mathscr{T}$ traversal, `dynMaxWeight` $(\cdot)$ will be invoked to incrementally compute maximum weight matching $M(*, X)$. Thus, the time of computing $M(*, X)$ can be significantly reduced from $O(d^3)$ to $O(d^2 w)$ with $w = W(*, X) \leq d$.

(2) Time per Phase. Fig. 4b depicts the time allocated in each phase of RS$^+$, *i.e.,* I) Preprocessing, and II) Iterative Shared Computing, on real datasets. We see that, on each dataset, the time taken in Phase I is always far less than that in Phase II. This indicates that it is worthwhile sacrificing only a little time in preprocessing the Steiner tree to achieve a drastic speedup in subsequent iterations. This agrees well with the complexity bound.

(3) Pruning Power. Fig. 4c shows the number of pairs in the weight matrix W pruned by Pruning Rules 1–3, respectively, on real datasets. We notice that (i) Pruning Rule 1 is the most powerful on all datasets as it can discard a large number of unpromising pairs with lightweight cost; Rule 2 the second, as expected. (ii) The last column shows a huge percentage of the pruned pairs (*e.g.,* 50.8% of pairs pruned on HT) relative to all $|\mathscr{V}|^2$ pairs in $\mathscr{G}$, highlighting the effectiveness of our pruning approaches.

(4) Accuracy & Exactness. Fig. 4d compares the iterative error $\epsilon_K = \text{ave}_{(x,y)}|s_K(x, y) - s(x, y)|$ of RS$^+$ and RS, respectively, at each iteration, for each fixed $\beta \in \{0.4, 0.6, 0.8\}$ on GR. The trends on other datasets are similar, and are omitted here due to space limitations. We notice that (i) given β, each K-th iterative errors of RS$^+$ and RS are exactly the same, showing that our shared computing methods do not sacrifice any accuracy for achieving a decent speedup. (ii) Both iterative errors of RS$^+$ and RS decrease to 0 as K increases, indicating that our RS$^+$ has the same convergence rate as RS.

5 CONCLUSIONS

We provide an efficient algorithm to accelerate RoleSim computation without loss of accuracy. First, we leverage a Steiner tree to find an optimised topological order of nodes for computational sharing. Next, an efficient algorithm, RoleSim+, is proposed to speed up RoleSim computation based on Steiner tree traversal. Our evaluations on real datasets validate the efficiency of RS$^+$.

Acknowledgments. This work was supported by the National Natural Science Foundation of China (NSFC61972203), and Natural Science Foundation of Jiangsu Province (BK20190442).

REFERENCES

[1] Michael L Fredman and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)* 34, 3 (1987), 596–615.

[2] Glen Jeh and Jennifer Widom. 2002. SimRank: A measure of structural-context similarity. In *SIGKDD*. 538–543. https://doi.org/10.1145/775047.775126

[3] Ruoming Jin, Victor E Lee, and Hui Hong. 2011. Axiomatic ranking of network role similarity. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 922–930.

[4] Ruoming Jin, Victor E Lee, and Longjie Li. 2014. Scalable and axiomatic ranking of network role similarity. *ACM Transactions on Knowledge Discovery from Data (TKDD)* 8, 1 (2014), 1–37.

[5] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford Large Network Dataset Collection. http://snap.stanford.edu/data.

[6] Longjie Li, Lvjian Qian, Victor E Lee, Mingwei Leng, Mei Chen, and Xiaoyun Chen. 2015. Fast and accurate computation of role similarity via vertex centrality. In *International Conference on Web-Age Information Management*. Springer, 123–134.

[7] Zhenjiang Lin, Michael R Lyu, and Irwin King. 2009. Matchsim: a novel neighbor-based similarity measure with maximum neighborhood matching. In *Proceedings of the 18th ACM conference on Information and knowledge management*. 1613–1616.

[8] G Ayorkor Mills-Tettey, Anthony Stentz, and M Bernardine Dias. 2007. The dynamic hungarian algorithm for the assignment problem with changing costs. *Robotics Institute, Pittsburgh, PA, Tech. Rep. CMU-RI-TR-07-27* (2007).

[9] Sascha Rothe and Hinrich Schütze. 2014. CoSimRank: A Flexible & Efficient Graph-Theoretic Similarity Measure. In *ACL*. 1392–1402. http://aclweb.org/anthology/P/P14/P14-1131.pdf

[10] Yingxia Shao, Jialin Liu, Shuyang Shi, Yuemei Zhang, and Bin Cui. 2019. Fast De-anonymization of Social Networks with Structural Information. *Data Science and Engineering* 4, 1 (2019), 76–92.

[11] Ismail H. Toroslu and Göktürk Üçoluk. 2007. Incremental assignment problem. *Information Sciences* 177, 6 (2007), 1523–1529. https://doi.org/10.1016/j.ins.2006.05.004

[12] Wensi Xi, Edward A. Fox, Weiguo Fan, Benyu Zhang, Zheng Chen, Jun Yan, and Dong Zhuang. 2005. SimFusion: Measuring similarity using unified relationship matrix. In *SIGIR*.

[13] Seok-Ho Yoon, Sang-Wook Kim, and Sunju Park. 2016. C-Rank: A link-based similarity measure for scientific literature databases. *Inf. Sci.* 326 (2016), 25–40. https://doi.org/10.1016/j.ins.2015.07.036

[14] Weiren Yu, Sima Iranmanesh, Aparajita Haldar, Maoyin Zhang, and Hakan Ferhatosmanoglu. 2021. RoleSim*: Scaling axiomatic role-based similarity ranking on large graphs. *World Wide Web* (2021), 1–45.

[15] Weiren Yu, Xuemin Lin, Wenjie Zhang, Jian Pei, and Julie A. McCann. 2019. SimRank*: Effective and scalable pairwise similarity search based on graph topology. *VLDB J.* 28, 3 (2019), 401–426. https://doi.org/10.1007/s00778-018-0536-3

[16] Weiren Yu, Xuemin Lin, Wenjie Zhang, Ying Zhang, and Jiajin Le. 2012. SimFusion+: Extending SimFusion towards efficient estimation on large and dynamic networks. In *SIGIR*.

[17] Weiren Yu, Julie McCann, Chengyuan Zhang, and Hakan Ferhatosmanoglu. 2022. Scaling High-Quality Pairwise Link-Based Similarity Retrieval on Billion-Edge Graphs. *ACM Trans. Inf. Syst.* 40, 4, Article 78 (2022), 45 pages. https://doi.org/10.1145/3495209

[18] Weiren Yu and Julie A. McCann. 2016. Random Walk with Restart over Dynamic Graphs. In *ICDM*. 589–598. https://doi.org/10.1109/ICDM.2016.0070

[19] Weiren Yu, Julie A. McCann, and Chengyuan Zhang. 2019. Efficient Pairwise Penetrating-rank Similarity Retrieval. *ACM Trans. Web* 13, 4 (2019), 21:1–21:52. https://doi.org/10.1145/3368616

[20] Weiren Yu and Fan Wang. 2018. Fast Exact CoSimRank Search on Evolving and Static Graphs. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018*, Pierre-Antoine Champin, Fabien Gandon, Mounia Lalmas, and Panagiotis G. Ipeirotis (Eds.). ACM, 599–608. https://doi.org/10.1145/3178876.3186126

[21] Peixiang Zhao, Jiawei Han, and Yizhou Sun. 2009. P-Rank: A comprehensive structural similarity measure over information networks. In *CIKM*. 553–562. https://doi.org/10.1145/1645953.1646025