Fast Exact CoSimRank Search on Evolving and Static Graphs

Weiren Yu
Aston University
Birmingham, B4 7ET, UK
w.yu3@aston.ac.uk

Fan Wang
Aston University
Birmingham, B4 7ET, UK
wangf7@aston.ac.uk

ABSTRACT

In real Web applications, CoSimRank has been proposed as a powerful measure of node-pair similarity based on graph topologies. However, existing work on CoSimRank is restricted to static graphs. When the graph is updated with new edges arriving over time, it is cost-inhibitive to recompute all CoSimRank scores from scratch, which is impractical. In this study, we propose a fast dynamic scheme, D-CoSim, for accurate CoSimRank search over evolving graphs. Based on D-CoSim, we also propose a fast scheme, F-CoSim, that greatly accelerates CoSimRank search over static graphs. Our theoretical analysis shows that D-CoSim and F-CoSim guarantee the exactness of CoSimRank scores. On the static graph G, to efficiently retrieve CoSimRank scores $\mathbf{S}$, F-CoSim is based on three ideas: (i) It first finds a "spanning polytree" T over G. (ii) On T, a fast algorithm is designed to compute the CoSimRank scores $\mathbf{S}(T)$ over the "spanning polytree" T. (iii) On G, D-CoSim is employed to compute the changes of $\mathbf{S}(T)$ in response to the delta graph $(G \ominus T)$. Experimental evaluations verify the superiority of D-CoSim over evolving graphs, and the fast speedup of F-CoSim on large-scale static graphs against its competitors, without any loss of accuracy.

1 INTRODUCTION

Graphs are widely used to model complex objects (*e.g.,* web pages) and their relationships (*e.g.,* hyperlinks). CoSimRank, proposed by Rothe and Schütze [18], is a powerful similarity measure between two objects based on graph topologies. It recursively follows the SimRank-like philosophy that "two nodes are considered as similar if their in-neighbours are similar". CoSimRank is a *node-pair* similarity measure, which is different from PageRank that ranks *nodes* only. Intuitively, a CoSimRank score $s(a, b)$ between nodes a and b aggregates *all* the meeting time of two random surfers starting at a and b, in contrast to SimRank [8] that counts their *first* meeting time only. Thus, CoSimRank has been shown [18] to be more accurate and effective than SimRank in many applications.

Application 1 (Synonym Expansion). Synonym expansion is a useful tool in search engine query rewriting [2, 5] and text simplification [4] that replaces a target word in a sentence with another more appropriate word. The CoSimRank measure was utilised to measure the similarity of words based on the intuition that "two words that are synonyms of each other should have similar lexical neighbors", where nodes are nouns, adjective and verbs occurring in Wikipedia, and edges denote types of syntactic configurations extracted from the parsed Wikipedias (*e.g.,* adjective-noun, verb-object, and noun-noun coordination). They evaluated the CoSimRank similarities of words (synonyms), whose results are superior to the cosine similarity of two Personalised PageRank vectors to identify effective synonyms.

Application 2 (Lexicon Extraction). Automatically building bilingual lexicons from corpora is an important task in natural language processing. Rothe and Schütze have applied CoSimRank to lexicon extraction, and represented an English and a German text corpus as two graphs, where nodes represent words, and edges denote grammatical relationships between words. Their central intuition is that "a node in the English graph and a node in the German graph are similar (*i.e.,* are likely to be translations of each other) if their neighbouring nodes are similar". They initialised the CoSimRank scores using an English-German "seed" dictionary whose entries correspond to known pairs of equivalent nodes (words). Their approach produces more reliable similarity results than SimRank-based approaches [11, 22].

Despite its effectiveness, existing work on CoSimRank is restricted to static graphs. However, when the graph is updated with new edges arriving over time, it is difficult for this approach to handle quick response over dynamical graphs, due to its cost-inhibitive overheads for recomputing CoSimRank scores from scratch. This highlights our need to consider the problem of fast accurate dynamic CoSimRank search:

Problem 1 (Dynamic CoSimRank on Evolving Graphs).

Given:	a graph G, a collection of edge updates ΔG to G, and a query set $Q = \{q_1, q_2, \cdots\}$.
Retrieve:	the changes to the CoSimRank scores *w.r.t.* Q on $(G \oplus \Delta G)$ quickly and accurately.

To address this issue, we propose a fast accurate dynamic scheme, D-CoSim, for CoSimRank search over evolving graphs. Moreover, as an important application of D-CoSim, we show that our dynamic D-CoSim is also applicable to static graphs to achieve a huge speedup for large-scale CoSimRank search. Thus, based on D-CoSim, we also design a fast accurate static scheme, F-CoSim, to solve the following problem:

Problem 2 (Static CoSimRank on Large Graphs).

Given:	a graph G, and a query set $Q = \{q_1, q_2, \cdots\}$
Retrieve:	the CoSimRank scores *w.r.t.* Q on G quickly and accurately.

To speed up the computation of CoSimRank scores $\mathbf{S}$ over the static graph G, (i) F-CoSim first finds a "spanning polytree" T over G; (ii) on the "spanning polytree" T, we devise a fast approach to compute the CoSimRank scores $\mathbf{S}(T)$ of T; (iii) on $(G \ominus T)$, we employ D-CoSim to compute the changes of $\mathbf{S}(T)$ *w.r.t.* the delta graph $(G \ominus T)$. With these ideas, F-CoSim and D-CoSim have the following salient features:

- **Fast.** F-CoSim and D-CoSim are orders of magnitude faster than the best-known competitors on static and dynamic graphs, respectively, with no loss of accuracy.
- **Dynamic.** D-CoSim quickly and accurately answers ad-hoc CoSimRank search on evolving graphs, with no need to recompute CoSimRank scores from scratch.
- **Accurate.** F-CoSim and D-CoSim do not compromise any accuracy for huge speedup.
- **Scalable.** Our schemes require only linear memory space, and scales well on million-node graphs.

In a nutshell, both dynamic D-CoSim and static F-CoSim allow myriads of SimRank-based applications [6, 14, 21, 30] being handled more efficiently and accurately.

2 RELATED WORK

Previous work on CoSimRank search focuses on static graphs. The pioneering research of [18] proposed an efficient local algorithm that computes each CoSimRank score from the sum of the dot product of two Personalised PageRank vectors. It entails $O(Kdn)$ time and $O(dn)$ memory to compute a single-pair CoSimRank score over a static graph with n nodes and d average degree after K iterations. However, when the graph is slightly updated, all CoSimRank scores have to be recomputed from scratch. Recently, Yu and McCann [27] have suggested an optimisation technique, namely CoSimMate, that leverages repeated squaring memoisation to cut down the number of iterations from K to $\lceil \log_2 K \rceil$ for all-pairs CoSimRank scores retrieval, but this approach requires extra $O(n^2)$ memory to store repeated squaring results, which is impractical on large-scale graphs. Worse still, the approach of [27] is a non-local algorithm on static graphs, meaning that, even if one wishes to compute a single-pair score, all-pairs scores have to be computed simultaneously.

Regarding dynamic updating, there is no work on CoSimRank except a relatively little work on updating of SimRank, a variant of SimRank, in dynamic graphs [9, 13, 20, 25, 26]. However, when extended to CoSimRank, these work would become inefficient, due to the following reasons: First, the two state-of-art studies [9, 20] are based on random walk sampling, whose optimisation techniques heavily hinge on aggregating "only the *first* meeting time" of two random surfers for SimRank. If applied to aggregate "*all* the meeting time" of two random surfers for CoSimRank, their approaches will become slow, due to the expensive cost to sample more additional meeting paths of two coalescing random walks. Second, some work [13, 25] devised low-rank decomposition methods to update all-pairs SimRank scores, leading to $O(n^2)$ memory to store the decomposed matrices, which is not scalable on large graphs. Worse still, these methods rest on an assumption that *all pairs* of old SimRank scores should be given in advance even if only a few pairs of scores need updating, which is unrealistic in practice.

Symbol	Description
G	given (old) graph G
ΔG	update graph to (old) graph G
$n/\tilde{n}$	number of nodes in old/new graph
$m/\tilde{m}$	number of edges in old/new graph
$\deg_i^-$	in-degree of node i in (old) graph G
C	damping factor $(0 < C < 1)$
K	number of iterations
$\mathbf{A}/\tilde{\mathbf{A}}$	old/new column-normalised adjacency matrix
$\mathbf{S}/\tilde{\mathbf{S}}$	old/new SimRank matrix
$\mathbf{I}$	$n \times n$ identity matrix
$\mathbf{e}_i$	$n \times 1$ unit vector with only a 1 in i-th entry
$\mathbf{X}^T$	transpose of matrix $\mathbf{X}$
$\mathbf{X}[i,:]$	i-th row of matrix $\mathbf{X}$
$\mathbf{X}[:,j]$	j-th column of matrix $\mathbf{X}$
$\mathbf{X}[i,j]$	(i,j)-th entry of matrix $\mathbf{X}$

Table 1: Description of Main Symbols

There is also a growing body of research on SimRank (the variant of CoSimRank) on static graphs [6–8, 10, 13, 15, 16, 19, 23, 29]. Their optimisation techniques can be classified into three broad categories: Monte Carlo sampling [6, 10, 19, 23], matrix-based methods [7, 13], and iterative schemes [8, 15, 29]. Among them, the sampling approach, SLING [23], is the best-of-breed SimRank algorithm on static graphs. However, their techniques, if applied to CoSimRank, are not fast as the performance gain of SLING relies on aggregating only the *first* meeting time of two coalescing walks, as opposed to CoSimRank that aggregates *all* their meeting time.

There has also been much work on computing incremental Personalised PageRank (PPR) vectors [3], and dynamic Random Walk with Restart (RWR) proximities [28]. However, it is not efficient to directly apply these techniques to dynamic CoSimRank updating. This is because the CoSimRank score at iteration k is the sum of k inner products between two Personalised PageRank vectors at every iteration $i = 1, 2, \cdots, k$. Thus, to update the k-th iterative CoSimRank score, existing incremental PPR (RWR) algorithms will be repeatedly applied $2k$ times to update two PPR (RWR) vectors at every iteration $i = 1, 2, \cdots, k$, respectively, before summing up the k dot products of every two PPR (RWR) vectors at each iteration, which would become rather expensive.

3 PRELIMINARIES

Let us formally revisit the CoSimRank definition. Table 1 lists the main notations used throughout this paper. CoSimRank, proposed by [18], is an attractive node-pair similarity measure based on graph topologies. It is based on a recursive philosophy that "two nodes are considered as similar if their in-neighbours are similar". Unlike SimRank [8], the CoSimRank score of each node with itself is not constantly 1. Mathematically, CoSimRank is formulated as follows:[1]

$$\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I} \tag{1}$$

[1]In comparison, SimRank [8] is defined as: $\mathbf{S} = \max\{C\mathbf{A}^T\mathbf{S}\mathbf{A}, \mathbf{I}\}$.

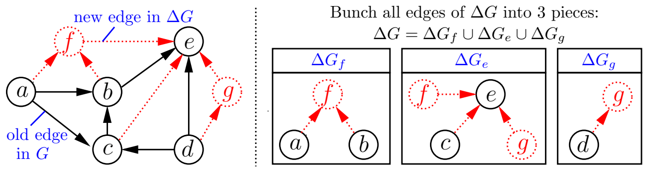

new edge in ΔG
old edge in G
Bunch all edges of ΔG into 3 pieces:
$\Delta G = \Delta G_f \cup \Delta G_e \cup \Delta G_g$
ΔG_f
ΔG_e
ΔG_g
a
b
c
d
e
f
g

Figure 1: Example of old web graph G (solid arrows) updated by ΔG (six new edges in dashed arrows)

where $\mathbf{S}$ is a (symmetric) CoSimRank matrix, whose element $\mathbf{S}[i,j]$ is the similarity score between nodes i and j in the graph G; C is a constant decay factor between 0 and 1; $\mathbf{A}$ is the column-normalised adjacency matrix; $\mathbf{I}$ is the identity matrix; and $(*)^T$ is matrix transpose. The upper bound of each element of $\mathbf{S}$ in Eq.(1) is $1/(1-C)$.

To evaluate one single-pair CoSimRank score, Rothe and Schütze [18] adopted a novel method to compute $\mathbf{S}[i,j]$:

$$\mathbf{S}[i,j] = \textstyle\sum_{k=0}^{\infty} C^k (\mathbf{p}_i^{(k)})^T \mathbf{p}_j^{(k)} \tag{2}$$

where $\mathbf{p}_j^{(k)}$ is the Personalised PageRank vector with respect to the seed node j, which can be iteratively obtained from

$$\mathbf{p}_j^{(k)} = \mathbf{A}\mathbf{p}_j^{(k-1)} \quad \text{with} \quad \mathbf{p}_j^{(0)} = \mathbf{e}_j \tag{3}$$

It requires $O(K(m+n))$ time and $O(m+Kn)$ memory to compute a single-pair score $\mathbf{S}[i,j]$ via Eqs.(2) and (3) on the static graph G with n nodes and m edges after K iterations. When the graph is dynamically updated, it will incur expensive cost to recompute all CoSimRank scores from scratch.

4 PROPOSED SCHEMES

We first present our efficient dynamic scheme, D-CoSim, that can quickly and accurately retrieve CoSimRank scores over large evolving graphs. Next, we will show that our dynamic D-CoSim is applicable to greatly speed up CoSimRank search over static graphs, and propose our static scheme, F-CoSim.

4.1 D-CoSim over Evolving Graphs

Given an old graph G, and a set of new edges updated to G:

$$\Delta G = \{(v_1 \to u_1), (v_2 \to u_2), (v_3 \to u_3), \cdots\}$$

According to the end point u_i of each edge $(v_i \to u_i)$ in ΔG, we first bunch all edges in ΔG into pieces:

$$\Delta G = \Delta G_{u_1} \cup \Delta G_{u_2} \cup \cdots \cup \Delta G_{u_p}$$

such that all edges in each piece ΔG_{u_i} share a common end point u_i. Thus, each piece ΔG_{u_i} takes the following form:

$$\Delta G_{u_i} = \{(v_{i_1} \to u_i), (v_{i_2} \to u_i), \cdots, (v_{i_\delta} \to u_i)\}$$

which is abbreviated as $\Delta G_{u_i} \triangleq ([v_{i_1}, v_{i_2}, \cdots, v_{i_\delta}] \to u_i)$.

Example 1. *Figure 1 depicts old graph G (solid arrows), and an update graph ΔG (dashed arrows) to G:*

$$\Delta G = \{(a \to f), (c \to e), (d \to g), (f \to e), (b \to f), (g \to e)\}.$$

We lump edges of ΔG into 3 pieces: $\Delta G = \Delta G_e \cup \Delta G_f \cup \Delta G_g$, where $\Delta G_e = \{(c \to e), (f \to e), (g \to e)\} \triangleq ([c,f,g] \to e)$,

$$\Delta G_f = \{(a \to f), (b \to f)\} \triangleq ([a,b] \to f), \quad \Delta G_g = ([d] \to g). \quad \square$$

The way we chunk edges of ΔG has two advantages: First, we can efficiently characterise the changes to $\mathbf{A}$ in answer to ΔG_{u_i} as a linear transformation of the u_i-th column of the old $\mathbf{A}$. This characterisation allows us to dynamically capture only the "refreshed areas" of CoSimRank scores in answer to the update ΔG_{u_i}. Second, bunching edges of ΔG facilitates sharing and reuse of common information among all the edge updates over each piece ΔG_{u_i}, thus discarding many unnecessary repeated computations on evolving graphs. For instance, to efficiently update CoSimRank similarities in response to each piece $\Delta G_{u_i} \triangleq ([v_{i_1}, v_{i_2}, \cdots, v_{i_\delta}] \to u_i)$, the intermediate results to update the edge $(v_{i_1} \to u_i)$, once computed, can be maximally reused to update all the other edges (*e.g.*, $(v_{i_2} \to u_i), (v_{i_3} \to u_i), \cdots$) in ΔG_{u_i}. Therefore, D-CoSim is highly efficient over evolving graphs.

Having bunched all edges of ΔG into chunks, we propose an efficient approach that dynamically computes the changes to the CoSimRank scores in response to each update piece ΔG_u.[2] We observe that each update piece ΔG_u changes only one column of $\mathbf{A}$. Specifically, we show the following lemma.

Lemma 1. *Given old graph G, and an update piece to G: $\Delta G_u = ([v_1, v_2, \cdots, v_{\delta_u}] \to u)$, the new column-normalised adjacency matrix $\tilde{\mathbf{A}}$ of the graph $(G \oplus \Delta G_u)$ can be dynamically updated from old $\mathbf{A}$ by replacing its u-th column with*

$$\tilde{\mathbf{A}}[:,u] = \tfrac{1}{\delta_u + \deg_u^-} \left(\deg_u^- \mathbf{A}[:,u] + \mathbf{1}_{\{v_1, v_2, \cdots, v_{\delta_u}\}} \right) \tag{4}$$

where $\deg_u^-$ is the in-degree of node u in old graph G; δ_u is the number of edge updates in ΔG_u; and $\mathbf{1}_{\{v_1, v_2, \cdots, v_{\delta_u}\}}$ is a column vector (its length is the number of rows in new $\tilde{\mathbf{A}}$) with 1s in the $(v_1, v_2, \cdots, v_{\delta_u})$-th entries, and 0s elsewhere.

Note that if the new $\tilde{\mathbf{A}}$ and old $\mathbf{A}$ are not of the same size (this case will happen when there are new nodes in ΔG_u), then prior to using Eq.(4), we should first border $\mathbf{A}$ with new zero-columns on the right and new zero-rows on the bottom to make it the same size of new $\tilde{\mathbf{A}}$.

Example 2. *In Figure 1, old graph G has 5 nodes, so the old $\mathbf{A}$ is of size 5×5. In $\Delta G_e = ([c,f,g] \to e)$ there are two new nodes f and g. Thus, to update $\mathbf{A}$ in answer to ΔG_e, we first border $\mathbf{A}$ to 7×7 with two zero columns and rows:*

$$\overset{\text{(bordered) old }\mathbf{A}}{\begin{array}{c|ccccccc|} & a & b & c & d & e & f & g \\ \hline a & 0 & \frac12 & \frac12 & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 0 & 0 & \frac12 & 0 & 0 \\ c & 0 & \frac12 & 0 & 0 & 0 & 0 & 0 \\ d & 0 & 0 & \frac12 & 0 & \frac12 & 0 & 0 \\ e & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ f & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ g & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}} \; \begin{array}{c}\text{bordered}\\\text{region}\end{array} \quad \underbrace{\tfrac{1}{3+2}}_{\delta_e \;\; \deg_e^-} \times \left(2 \times \overset{\text{old }\mathbf{A}[:,e]}{\begin{array}{c|c|} a & 0 \\ b & \frac12 \\ c & 0 \\ d & \frac12 \\ e & 0 \\ f & 0 \\ g & 0 \end{array}} + \overset{\mathbf{1}_{\{c,f,g\}}}{\begin{array}{c|c|} a & 0 \\ b & 0 \\ c & 1 \\ d & 0 \\ e & 0 \\ f & 1 \\ g & 1 \end{array}} \right) = \overset{\text{new }\tilde{\mathbf{A}}[:,e]}{\begin{array}{c|c|} a & 0 \\ b & \frac15 \\ c & \frac15 \\ d & \frac15 \\ e & 0 \\ f & \frac15 \\ g & \frac15 \end{array}}$$

Then, since $\deg_e^- = 2$ and $\delta_e = 3$, in light of Eq.(3), the e-th column of $\mathbf{A}$ in answer to ΔG_e is updated to

$$\tilde{\mathbf{A}}[:,e] = \tfrac{1}{3+2} \left(2\mathbf{A}[:,e] + \mathbf{1}_{\{c,f,g\}} \right) = [0, \tfrac15, \tfrac15, \tfrac15, 0, \tfrac15, \tfrac15]^T. \quad \square$$

[2] In the following, $\Delta G_{u_i} = ([v_{i_1}, v_{i_2}, \cdots, v_{i_\delta}] \to u_i)$ is abbreviated to $\Delta G_u = ([v_1, v_2, \cdots, v_{\delta_u}] \to u)$ for simplicity.

Leveraging Lemma 1, we next show how to dynamically update CoSimRank scores in answer to each piece ΔG_u.

Theorem 1. *Given an old graph G, an update piece to G: $\Delta G_u = ([v_1, \cdots, v_{\delta_u}] \to u)$, and a query node $q \in (G \oplus \Delta G_u)$, the changes $\mathbf{\Delta S}[:, q]$ to CoSimRank scores with respect to q are dynamically computed as*

$$\mathbf{\Delta S}[:, q] = \sum_{k=0}^{\infty} C^k \left(\mathbf{t}^{(k)}[q] \cdot \mathbf{p}^{(k)} + \mathbf{p}^{(k)}[q] \cdot \mathbf{t}^{(k)} \right) \quad (5)$$

where $\mathbf{p}^{(k)}[q]$ and $\mathbf{t}^{(k)}[q]$ denote the q-th entry of the vectors $\mathbf{p}^{(k)}$ and $\mathbf{t}^{(k)}$, respectively, which are iteratively obtained by

$$\begin{cases} \mathbf{p}^{(0)} = \mathbf{e}_u \\ \mathbf{p}^{(k)} = \tilde{\mathbf{A}}^T \mathbf{p}^{(k-1)} \end{cases} \begin{cases} \mathbf{t}^{(0)} = \frac{C}{2(\delta_u + \deg_u^-)} (\mathbf{A} + \tilde{\mathbf{A}})^T \mathbf{r} \\ \mathbf{t}^{(k)} = \tilde{\mathbf{A}}^T \mathbf{t}^{(k-1)} \end{cases} \quad (6)$$

and $\mathbf{r} = \lim_{K\to\infty} \mathbf{r}^{(K)}$, which can be iteratively derived as

$$\begin{cases} \mathbf{r}^{(0)} = \mathbf{w}^{(K)} \\ \mathbf{r}^{(k)} = C\mathbf{A}^T \mathbf{r}^{(k-1)} + \mathbf{w}^{(K-k)} \end{cases} \qquad (1 \le k \le K) \quad (7)$$

$$\textit{with} \begin{cases} \mathbf{w}^{(0)} = \mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u] \\ \mathbf{w}^{(k)} = \mathbf{A}\mathbf{w}^{(k-1)} \end{cases} \qquad (1 \le k \le K) \quad (8)$$

Proof. After ΔG_u is updated to G, by definition in Eq.(1), the new CoSimRank scores $(\mathbf{S} + \mathbf{\Delta S})$ in $G \oplus \Delta G_u$ satisfy

$$\mathbf{S} + \mathbf{\Delta S} = C\tilde{\mathbf{A}}^T (\mathbf{S} + \mathbf{\Delta S}) \tilde{\mathbf{A}} + \mathbf{I}$$

Rearranging the terms in the above equation yields

$$\mathbf{\Delta S} = C\tilde{\mathbf{A}}^T \mathbf{\Delta S} \tilde{\mathbf{A}} + \mathbf{E} \quad \text{with} \quad \mathbf{E} = C\tilde{\mathbf{A}}^T \mathbf{S} \tilde{\mathbf{A}} + \mathbf{I} - \mathbf{S} \quad (9)$$

Let $\mathbf{\Delta A} = \tilde{\mathbf{A}} - \mathbf{A}$. From Eq.(4) in Lemma 1, we have

$$\mathbf{\Delta A}[:, u] = \tfrac{1}{\delta_u + \deg_u^-} \left(\mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u] \right) \quad (10)$$

To simplify $\mathbf{E}$ in Eq.(9), we plug $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{\Delta A}[:, u]\mathbf{e}_u^T$ and $\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I}$, and let $\mathbf{f}_u \triangleq \mathbf{S}\mathbf{\Delta A}[:, u]$, which produces

$$\begin{aligned} \mathbf{E} &= C(\mathbf{e}_u(\mathbf{f}_u^T \mathbf{A}) + (\mathbf{A}^T \mathbf{f}_u)\mathbf{e}_u^T + (\mathbf{\Delta A}[:, u]^T \mathbf{f}_u)\mathbf{e}_u \mathbf{e}_u^T) \\ &= \mathbf{e}_u \mathbf{x}^T + \mathbf{x}\mathbf{e}_u^T \quad \text{where} \qquad (11) \\ \mathbf{x} &= C\mathbf{A}^T \mathbf{f}_u + \tfrac{C}{2}(\mathbf{\Delta A}[:, u]^T \mathbf{f}_u)\mathbf{e}_u \\ &= C(\mathbf{A}^T + \tfrac{1}{2}\mathbf{e}_u \mathbf{\Delta A}[:, u]^T)\mathbf{f}_u \qquad \{\text{using Eq.(10)}\} \\ &= \tfrac{C}{2}(\mathbf{A}^T + \tilde{\mathbf{A}}^T)\mathbf{f}_u \qquad (12) \end{aligned}$$

Thus, combining Eqs.(9) and (11), we obtain

$$\begin{aligned} \mathbf{\Delta S} &= \sum_{k=0}^{\infty} C^k (\tilde{\mathbf{A}}^T)^k \mathbf{E} \tilde{\mathbf{A}}^k \qquad \{\text{using Eq.(11)}\} \\ &= \sum_{k=0}^{\infty} C^k \left((\tilde{\mathbf{A}}^T)^k \mathbf{e}_u \mathbf{x}^T \tilde{\mathbf{A}}^k + (\tilde{\mathbf{A}}^T)^k \mathbf{x} \mathbf{e}_u^T \tilde{\mathbf{A}}^k \right) \qquad (13) \end{aligned}$$

By direct iteration, it follows from Eqs.(6) and (8) that

$$\mathbf{p}^{(k)} = (\tilde{\mathbf{A}}^T)^k \mathbf{e}_u, \quad \mathbf{w}^{(k)} = \mathbf{A}^k (\mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u]) \quad (14)$$

To express $\mathbf{r}$ in $\mathbf{t}^{(0)}$ of Eq.(6), by iteration, Eq.(7) implies

$$\mathbf{r}^{(K)} = \left(C\mathbf{A}^T \right)^K \mathbf{w}^{(K)} + \left(C\mathbf{A}^T \right)^{K-1} \mathbf{w}^{(K-1)} + \cdots + \mathbf{w}^{(0)}$$

$$= \underbrace{((C\mathbf{A}^T)^K \mathbf{A}^K + (C\mathbf{A}^T)^{K-1} \mathbf{A}^{K-1} + \cdots + \mathbf{I})}_{=\{\text{the first } K \text{ terms of } \mathbf{S} = \sum_{k=0}^{\infty} C^k (\mathbf{A}^T)^k \mathbf{A}^k\}} \underbrace{(\mathbf{1}_{\{v_1, \cdots, v_{\delta_u}\}} - \delta_u \mathbf{A}[:, u])}_{\{\text{By Eq.(10)}\} = (\delta_u + \deg_u^-)\mathbf{\Delta A}[:, u]}$$

By taking limits on both sides, we have

$$\mathbf{r} \triangleq \lim_{K\to\infty} \mathbf{r}^{(K)} = \left(\delta_u + \deg_u^- \right) \mathbf{S}\mathbf{\Delta A}[:, u] = \left(\delta_u + \deg_u^- \right) \mathbf{f}_u$$

Thus, by plugging $\mathbf{r} = \left(\delta_u + \deg_u^- \right) \mathbf{f}_u$ into Eq.(6), we get

$$\mathbf{t}^{(0)} = \tfrac{C}{2}(\mathbf{A} + \tilde{\mathbf{A}})^T \mathbf{f}_u = \{\text{By Eq.(12)}\} = \mathbf{x}, \quad \mathbf{t}^{(k)} = (\tilde{\mathbf{A}}^k)^T \mathbf{x} \quad (15)$$

Substituting Eqs.(14) and (15) into Eq.(13) produces

$$\mathbf{\Delta S} = \sum_{k=0}^{\infty} C^k \left(\mathbf{p}^{(k)} \left(\mathbf{t}^{(k)} \right)^T + \mathbf{t}^{(k)} \left(\mathbf{p}^{(k)} \right)^T \right)$$

Finally, post-multiplying both sides by $\mathbf{e}_q$ yields Eq.(5). □

Example 3. *Recall the old G (solid arrows) in Figure 1, and update piece $\Delta G_e = ([c, f, g] \to e)$ to G (dashed arrows). Given query $q = e$, number of iterations $K = 3$, and decay factor $C = 0.6$, Theorem 1 retrieves $\mathbf{\Delta S}[:, e]$ as follows:*

First, we compute $\{\mathbf{w}^{(k)}\}$ and $\{\mathbf{r}^{(k)}\}$ via Eqs.(8) and (7):

k	$\mathbf{w}^{(k)}$	$\mathbf{r}^{(k)}$
0	$[0, -1.5, 1, -1.5, 0, 1, 1]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$
1	$[-.25, 0, -.75, .5, 0, 0, 0]^T$	$[-.375, 0, 0, -.375, 0, 0, 0]^T$
2	$[-.375, 0, 0, -.375, 0, 0, 0]^T$	$[-.25, -.113, -.975, .5, -.113, 0, 0]^T$
3	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, -1.868, 1.075, -1.5, .116, 1, 1]^T$

Next, we obtain $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ via Eq.(6) with $\mathbf{r} = \mathbf{r}^{(3)}$:

k	$\mathbf{p}^{(k)}$	$\mathbf{t}^{(k)}$
0	$[0, 0, 0, 0, 1, 0, 0]^T$	$[0, .065, -.09, 0, -.105, 0, 0]^T$
1	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, -.045, 0, 0, -.005, 0, 0]^T$
2	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, -.009, 0, 0]^T$
3	$[0, 0, 0, 0, 0, 0, 0]^T$	$[0, 0, 0, 0, 0, 0, 0]^T$

Finally, we use Eq.(5) to derive $\mathbf{\Delta S}[:, e]$ in answer to ΔG_e:

$$\begin{aligned} \mathbf{\Delta S}[:, e] &= \textstyle\sum_{k=0}^{3} 0.6^k \left(\mathbf{t}^{(k)}[e] \cdot \mathbf{p}^{(k)} + \mathbf{p}^{(k)}[e] \cdot \mathbf{t}^{(k)} \right) \\ &= [0, .0645, -.09, 0, -.2091, 0, 0]^T \quad \square \end{aligned}$$

Theorem 1 implies an efficient dynamic method, D-CoSim, to retrieve the changes to CoSimRank scores (Algorithm 1).

Correctness. While Theorem 1 guarantees the correctness of $\mathbf{\Delta S}$ *w.r.t. one piece* update ΔG_u only, the following theorem guarantees further that, after one piece update ΔG_u is processed, other pieces being processed will not distort the correct CoSimRank results $\mathbf{\Delta S}[:, Q]$.

Theorem 2. *Let $\Delta G \triangleq \{\Delta G_1, \Delta G_2, \cdots, \Delta G_p\}$ be a set of edges bunched into pieces updated to the old graph G (line 1). The CoSimRank changes $\mathbf{\Delta S}$ (line 20) returned by D-CoSim are the correct answers in response to the update graph ΔG.*

Proof. Let $\mathbf{\Delta A}, \mathbf{\Delta A_1}, \mathbf{\Delta A_2}, \cdots, \mathbf{\Delta A_p}$ be the changes to the column-normalised adjacency matrices *w.r.t.* the graph updates $\Delta G, \Delta G_1, \Delta G_2, \cdots, \Delta G_p$, respectively.

In the 1st round of for-loop (lines 4–19): D-CoSim starts by viewing G_0 ($\triangleq G$) as the old graph, and $\mathbf{S_0}$ ($\triangleq \mathbf{S}$) as the old CoSimRank scores, and update the 1st chunk ΔG_1 to G_0. Theorem 1 ensures that $\mathbf{s}$ (line 18) at the 1st round, denoted by $\mathbf{\Delta S_1}$, is the CoSimRank changes *w.r.t.* the update ΔG_1 to G_0, *i.e.*, $\mathbf{\Delta S_1}$ satisfies

$$\mathbf{S_0} + \mathbf{\Delta S_1} = C(\mathbf{A_0} + \mathbf{\Delta A_1})^T (\mathbf{S_0} + \mathbf{\Delta S_1})(\mathbf{A_0} + \mathbf{\Delta A_1}) + \mathbf{I}$$

Algorithm 1: D-CoSim $(G, \Delta G, C, Q, K)$

```
Input : an old graph $G$, a set of edge updates $\Delta G$ to $G$,
        decay factor $C$, a query set $Q$, #-iteration $K$
Output: CoSimRank changes $\mathbf{\Delta S}[:,Q]$ in answer to $\Delta G$.
1  chunk all edges of $\Delta G$ to $\{\Delta G_u\}$ s.t. $\Delta G = \bigcup_u \Delta G_u$ and
   edges in $\Delta G_u = ([v_1, \cdots, v_{\delta_u}] \to u)$ share common end $u$
2  foreach query $q \in Q$ do initialise $\mathbf{\Delta S}[:,q] := \mathbf{0}$
3  foreach piece $\Delta G_u$ do
4      initialise $\mathbf{w}^{(0)} := \mathbf{1}_{\{v_1,\cdots,v_{\delta_u}\}} - \delta_u \mathbf{A}[:,u]$
5      for $k = 1$ to $K$ do update $\mathbf{w}^{(k)} := \mathbf{A}\mathbf{w}^{(k-1)}$
6      initialise $\mathbf{r} := \mathbf{w}^{(K)}$
7      for $k = 1$ to $K$ do update $\mathbf{r} := C\mathbf{A}^T\mathbf{r} + \mathbf{w}^{(K-k)}$
8      $\mathbf{\Delta A}[:,u] := \frac{1}{\delta_u + \deg_u^-}\left(\mathbf{1}_{\{v_1,\cdots,v_{\delta_u}\}} - \delta_u \mathbf{A}[:,u]\right)$
9      update $\tilde{\mathbf{A}} := \mathbf{A} + \mathbf{\Delta A}[:,u]\mathbf{e}_u^T$
10     initialise $\mathbf{p}^{(0)} := \mathbf{e}_u$
11     for $k = 1$ to $K$ do update $\mathbf{p}^{(k)} := \tilde{\mathbf{A}}^T\mathbf{p}^{(k-1)}$
12     initialise $\mathbf{t}^{(0)} := \frac{C}{2(\delta_u + \deg_u^-)}(\mathbf{A} + \tilde{\mathbf{A}})^T\mathbf{r}$
13     for $k = 1$ to $K$ do update $\mathbf{t}^{(k)} := \tilde{\mathbf{A}}\mathbf{t}^{(k-1)}$
14     foreach query $q \in Q$ do
15         initialise $\mathbf{s} := \mathbf{0}$;
16         for $k = 0$ to $K$ do
17             $\mathbf{s} := \mathbf{s} + C^k(\mathbf{t}^{(k)}[q] \cdot \mathbf{p}^{(k)} + \mathbf{p}^{(k)}[q] \cdot \mathbf{t}^{(k)})$;
18         update $\mathbf{\Delta S}[:,q] := \mathbf{\Delta S}[:,q] + \mathbf{s}$;
19     update $\mathbf{A} := \tilde{\mathbf{A}}$
20 return $\mathbf{\Delta S}[:,Q] := \{\mathbf{\Delta S}[:,q] \mid \forall q \in Q\}$;
```

Then, D-CoSim updates $\mathbf{\Delta S}$ (line 18) by adding $\mathbf{s}$ $(= \mathbf{\Delta S_1})$, and updates the current graph from G_0 to G_1 (line 19):

$$\mathbf{\Delta S} = \mathbf{0} + \mathbf{\Delta S_1} = \mathbf{\Delta S_1}, \qquad \mathbf{A_1} = \mathbf{A_0} + \mathbf{\Delta A_1}$$

The for-loop (lines 4–19) continues till the last chunk ΔG_p is updated. In the p-th (last) round of for-loop (lines 4–18): D-CoSim regards G_{p-1} $(= G_{p-2} + \Delta G_{p-1})$ as the old graph, and $\mathbf{S_{p-1}}(= \mathbf{S_{p-2}} + \mathbf{\Delta S_{p-1}})$ as the old CoSimRank scores, and updates the p-th chunk ΔG_p to G_{p-1}. Theorem 1 ensures that $\mathbf{s}$ (line 18) at the p-th round, denoted by $\mathbf{\Delta S_p}$, is the CoSimRank changes *w.r.t.* the update ΔG_p to G_{p-1}, *i.e.,* $\mathbf{\Delta S_p}$ satisfies

$$\mathbf{S_{p-1}} + \mathbf{\Delta S_p} = C(\mathbf{A_{p-1}} + \mathbf{\Delta A_p})^T \cdot (\mathbf{S_{p-1}} + \mathbf{\Delta S_p}) \cdot \cdot (\mathbf{A_{p-1}} + \mathbf{\Delta A_p}) + \mathbf{I} \tag{16}$$

Then, D-CoSim updates $\mathbf{\Delta S}$ (line 18) by adding $\mathbf{s}$ $(= \mathbf{\Delta S_p})$, and updates the current graph from G_{p-1} to G_p (line 19):

$$\mathbf{\Delta S} = (\mathbf{\Delta S_1} + \cdots + \mathbf{\Delta S_{p-1}}) + \mathbf{\Delta S_p}, \quad \mathbf{A_p} = \mathbf{A_{p-1}} + \mathbf{\Delta A_p}$$

Finally, we check if $\mathbf{\Delta S}$ $(= \mathbf{\Delta S_1} + \mathbf{\Delta S_2} + \cdots + \mathbf{\Delta S_p})$ is the correct CoSimRank changes *w.r.t.* the update ΔG to G. Our above analysis for each round of the for-loop implies

$$\mathbf{S_i} = \mathbf{S_{i-1}} + \mathbf{\Delta S_i}, \qquad \mathbf{A_i} = \mathbf{A_{i-1}} + \mathbf{\Delta A_i} \qquad (\forall i = 1, \cdots, p-1)$$

Repeatedly applying the above iterations produces

$$\begin{aligned}\mathbf{A_{p-1}} + \mathbf{\Delta A_p} &= (\mathbf{A_{p-2}} + \mathbf{\Delta A_{p-1}}) + \mathbf{\Delta A_p} = \cdots = \\ &= (\mathbf{A_0} + \mathbf{\Delta A_1}) + \mathbf{\Delta A_2} + \cdots + \mathbf{\Delta A_{p-1}} + \mathbf{\Delta A_p} \\ &= \mathbf{A_0} + \mathbf{\Delta A} \quad \text{with} \quad \mathbf{\Delta A} \triangleq \mathbf{\Delta A_1} + \cdots + \mathbf{\Delta A_p}\end{aligned} \tag{17}$$

Similarly,

$$\mathbf{S_{p-1}} + \mathbf{\Delta S_p} = \mathbf{S_0} + \mathbf{\Delta S} \;\; \text{with} \;\; \mathbf{\Delta S} \triangleq \mathbf{\Delta S_1} + \cdots + \mathbf{\Delta S_p} \tag{18}$$

Plugging Eqs.(17) and (18) into Eq.(16) produces

$$\mathbf{S_0} + \mathbf{\Delta S} = C(\mathbf{A_0} + \mathbf{\Delta A})^T(\mathbf{S_0} + \mathbf{\Delta S})(\mathbf{A_0} + \mathbf{\Delta A}) + \mathbf{I}$$

Thus, $\mathbf{\Delta S}$ satisfies the CoSimRank definition, which implies that $\mathbf{\Delta S}$ $(= \mathbf{\Delta S_1} + \mathbf{\Delta S_2} + \cdots + \mathbf{\Delta S_p})$ returned by D-CoSim is exactly the CoSimRank changes *w.r.t.* the graph update ΔG $(= \Delta G_1 + \cdots + \Delta G_p)$ to G_0 $(\triangleq G)$. □

EXAMPLE 4. *Recall old graph G (solid arrows) and update graph ΔG to G (dashed arrows) in Figure 1. Given the query $q = e$, number of iterations $K = 3$, and decay factor $C = 0.6$, D-CoSim computes $\mathbf{\Delta S}$ in answer to ΔG as follows:*

First, D-CoSim chunks all edges of ΔG into three pieces: $\Delta G = \Delta G_e \cup \Delta G_f \cup \Delta G_g$, according to Example 1.

Then, after CoSimRank changes $\mathbf{\Delta S_1}[:,e]$ w.r.t. 1st piece update ΔG_e to G_0 $(= G)$ are derived (see Example 3):

$$\mathbf{\Delta S_1}[:,e] = [0, .0645, -.09, 0, -.2091, 0, 0]^T,$$

D-CoSim views $G_1(= G_0 \oplus \Delta G_e)$ as the old graph, and computes changes $\mathbf{\Delta S_2}[:,e]$ w.r.t. 2nd piece update ΔG_f to G_1:

$$\mathbf{\Delta S_2}[:,e] = [0, .009, 0, 0, .0239, .1, 0]^T.$$

Next, it views $G_2(= G_1 \oplus \Delta G_g)$ as the old graph, and computes changes $\mathbf{\Delta S_3}[:,e]$ w.r.t. 3rd piece update ΔG_g to G_2:

$$\mathbf{\Delta S_3}[:,e] = [0, .018, 0, 0, .0288, 0, .12]^T.$$

Finally, the CoSimRank changes $\mathbf{\Delta S}[:,e]$ w.r.t. the graph update ΔG $(= \Delta G_e \oplus \Delta G_f \oplus \Delta G_g)$ are

$$\begin{aligned}\mathbf{\Delta S}[:,e] &= \mathbf{\Delta S_1}[:,e] + \mathbf{\Delta S_2}[:,e] + \mathbf{\Delta S_3}[:,e] \\ &= [0, .0915, -.09, 0, -.1564, .1, .12]^T\end{aligned} \qquad □$$

Complexity. We analyse the computational cost of D-CoSim. Let $\tilde{n}$ and $\tilde{m}$ denote the number of nodes and edges in new $G \oplus \Delta G$, respectively. Let δ be the number of edges in ΔG, and p be the number of update pieces $\{\Delta G_u\}$ in ΔG. Clearly, $p \leq \delta$. D-CoSim has the following complexity bound:

THEOREM 3. *D-CoSim requires $O(K(\tilde{m} + \tilde{n}p|Q|))$ time and $O(\tilde{m} + K\tilde{n})$ memory to dynamically compute $\mathbf{\Delta S}[:,Q]$ after K iterations, where $|Q|$ is the number of queries in Q.*

PROOF. D-CoSim runs in three phases: (1) bunching edges of ΔG (line 1), (2) $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ iterating (lines 4–13), and (3) online query (lines 14–18). Specifically, bunching edges of ΔG requires $O(\delta)$ time and $O(\delta)$ memory for a linear scan of all edges in ΔG. To iteratively compute $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$, for each query $q \in Q$ and each piece update ΔG_u, it entails $O(K\tilde{m})$ time and $O(\tilde{m} + K\tilde{n})$ memory for Eqs.(6)–(8). The $O(K\tilde{m})$ time is dominated by 5 matrix-vector products: $\mathbf{A}\mathbf{w}^{(k-1)}$ (line 5), $\mathbf{A}^T\mathbf{r}^{(k-1)}$ (line 7), $\tilde{\mathbf{A}}^T\mathbf{p}^{(k-1)}$ (line 11), $(\mathbf{A} + \tilde{\mathbf{A}})^T\mathbf{r}$ (line 12), and $\tilde{\mathbf{A}}\mathbf{t}^{(k-1)}$ (line 13). The memory $O(\tilde{m} + K\tilde{n})$ is dominated by the storage of matrix $\mathbf{A}$, and resulting iterative vectors. For online query, once $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ are computed, they are memoised and reused to compute $\mathbf{\Delta S}[:,q]$ for every query in Q. After $\mathbf{\Delta S}[:,q]$ is updated in answer to each piece ΔG_u, all the vectors $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$ are freed for the next piece update. Thus, for $|Q|$ queries and p update pieces, it entails $O(K(\tilde{m}+\tilde{n}p|Q|))$ time and $O(\tilde{m}+K\tilde{n})$ memory in total. □

Theorem 3 guarantees the high efficiency of D-CoSim for dynamic CoSimRank search, whose speedup is achieved by (a) our characterisation of the "refreshed areas" $\mathbf{\Delta S}[:,q]$ in terms of only the linear combination of $\{\mathbf{p}^{(k)}\}$ and $\{\mathbf{t}^{(k)}\}$, and (b) maximally reusing and sharing common intermediate results in answer to the edge updates on each piece ΔG_u. In comparison, the existing approach by [18] requires $O(K(\tilde{m}+\tilde{n}))$ time to compute only a single-pair $\tilde{\mathbf{S}}[i,j]$ per edge update via Eqs.(2) and (3) from scratch, leading to $O(K(\tilde{m}+\tilde{n}|Q|)\tilde{n}\delta)$ total time to compute $\tilde{\mathbf{S}}[:,Q]$ ($\tilde{n}\times|Q|$ pairs) for δ edge updates, which is rather expensive.

4.2 F-CoSim over Static Graphs

Apart from supporting quick dynamic CoSimRank retrieval on evolving graphs, D-CoSim can also be applied to static graphs for accelerating CoSimRank search. Based on D-CoSim, we next propose an efficient scheme, F-CoSim, that greatly speeds up CoSimRank search over static graphs. Given a static graph G and a query set Q, F-CoSim retrieves the CoSimRank scores $\mathbf{S}[:,Q]$ over G based on three ideas: First, we propose a fast method to find a "spanning polytree" T of G so that G is decomposed into $G=T\oplus(G\ominus T)$, which can be viewed as the old T plus its update $(G\ominus T)$. Next, on T, due to its special "polytree" structure, we notice that the CoSimRank scores are relatively easier to compute, and we propose a novel fast algorithm to retrieve the CoSimRank scores $\mathbf{S}(T)[:,Q]$ over the "spanning polytree". Finally, we apply our dynamic D-CoSim to compute $\mathbf{S}(T)$ changes in response to the graph update $(G\ominus T)$. With the above ideas, F-CoSim enables a notable speedup in CoSimRank search over static graphs, which is achieved by our efficient method to retrieve $\mathbf{S}(T)[:,Q]$ over the "spanning polytree" and our fast D-CoSim to compute the changes to $\mathbf{S}(T)$ *w.r.t.* $(G\ominus T)$.

In the following, we shall elaborate on these ideas.

Definition 1 (**Spanning Polytree**). *A spanning polytree T of a connected graph G is a subgraph of G that includes every node of G (i.e., spans G), with a maximal set of edges of G that contains no undirected cycles if we replace all the directed edges of T with undirected edges.*

Intuitively, in contrast with the traditional definition of the spanning tree in which each node has only one parent node, our spanning polytree is a generalised notion of the spanning tree from undirected graphs to directed ones, in which each node may have more than one parent nodes. The reason we introduce the spanning polytree is that, when G is a directed graph, its traditional spanning tree does not always exist, but there always exists a spanning polytree of G. For instance in Figure 2, there are no conventional trees that span G, but one can find a polytree T that spans G. If G is an undirected graph, the spanning polytree in Definition 1 reduces to the traditional spanning tree.

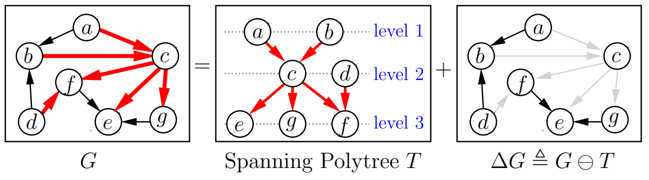


Figure 2: Decompose G into a spanning polytree T and ΔG $(=G\ominus T)$

To identify a spanning polytree T over a given graph G, we devise a fast heuristic approach based on breadth first search (BFS) in Procedure 1. The complexity of Procedure 1 is dominated by the BFS search, which is $O(n+m)$ time and $O(n+m)$ memory on a graph with n nodes and m edges.

```
Procedure 1: Find_Spanning_PolyTree (G)
  Input  : a graph $G$.
  Output : a spanning polytree $T$ of $G$.
1 initialise $T := \varnothing$ ;
2 foreach weakly connected component $G_{wcc} \subseteq G$ do
3     $U$.Enqueue(a node of maximum out-degree in $G_{\text{wcc}}$);
4     while $U \neq \varnothing$ do
5         set node $v := U$.Dequeue() ;
6         $T := T \cup \{$unvisited in- and out-links of $v\}$ ;
7         forall $w \in \{$unvisited in- and out-neighbors of $v\}$
          do $U$.Enqueue($w$);
8 return $T$ ;
```

Having identified the spanning polytree T of the graph G, we can decompose G into two parts: $G=T\oplus(G\ominus T)$. Due to the special acyclic structure of T, there is a more efficient way to retrieve CoSimRank scores of the spanning polytree T. Our key observation is that, if the nodes of T are organised in level order, the adjacency matrix $\mathbf{A}$ of T will exhibit a block superdiagonal structure, leading to the CoSimRank scores of T, $\mathbf{S}(T)$, displaying a block diagonal structure. Consequently, any two nodes at different levels of T have zero CoSimRank scores. Moreover, the CoSimRank scores of the nodes at the same level of T can be immediately derived from those at the previous level, based on the following theorem:

Theorem 4 (**CoSimRank on Polytree** T). *Given a polytree T with nodes organised in level order, let n_l be the number of nodes at level l $(l=1,\cdots,L)$, the CoSimRank scores of T, $\mathbf{S}(T)$, is computed level by level:*

$$\begin{aligned}\mathbf{S}(T)=\mathit{diag}(\mathbf{S}_1,\mathbf{S}_2,\cdots,\mathbf{S}_L)\quad &\textit{with}\quad \mathbf{S}_1=\mathbf{I}_{n_1}\quad \textit{and}\\ \mathbf{S}_l=C\mathbf{A}_{l-1,l}^T\mathbf{S}_{l-1}\mathbf{A}_{l-1,l}+\mathbf{I}_{n_l}&\quad (l=2,\cdots,L)\end{aligned}\tag{19}$$

where L is the number of levels in T; $\mathbf{S}(T)$ is a diagonal block matrix with each block $\mathbf{S}_l$ being the CoSimRank scores of n_l^2 pairs of nodes at level l; $\mathbf{A}_{l-1,l}$ is the $(n_{l-1}\times n_l)$ column-normalised adjacency matrix of the subgraph between level $(l-1)$ and level l of T; and $\mathbf{I}_{n_l}$ is the $n_l\times n_l$ identity matrix.

Proof. Since T is a polytree, two surfers starting at different levels cannot meet at a common node via equal-length steps. Thus, only node-pairs at the same level have nonzero scores, leading to the block diagonal structure of $\mathbf{S}(T)$.

To compute l-th diagonal block $\mathbf{S}_l$, by Eq.(1), we have

$$C\begin{bmatrix}\mathbf{0} & \mathbf{A}_{1,2} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \ddots & \vdots\\ \vdots & \vdots & \ddots & \mathbf{A}_{L-1,L}\\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix}^T \begin{bmatrix}\mathbf{S}_1 & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathbf{S}_2 & \cdots & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{S}_L\end{bmatrix} \begin{bmatrix}\mathbf{0} & \mathbf{A}_{1,2} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathbf{0} & \ddots & \vdots\\ \vdots & \vdots & \ddots & \mathbf{A}_{L-1,L}\\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0}\end{bmatrix} + \begin{bmatrix}\mathbf{I}_{n_1} & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathbf{I}_{n_2} & \cdots & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{I}_{n_L}\end{bmatrix}$$

$$= \begin{bmatrix}\mathbf{I}_{n_1} & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{0} & C\mathbf{A}_{1,2}^T\mathbf{S}_1\mathbf{A}_{1,2}+\mathbf{I}_{n_2} & \cdots & \mathbf{0}\\ \vdots & \ddots & \ddots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & C\mathbf{A}_{L-1,L}^T\mathbf{S}_{L-1}\mathbf{A}_{L-1,L}+\mathbf{I}_{n_L}\end{bmatrix} = \begin{bmatrix}\mathbf{S}_1 & \mathbf{0} & \cdots & \mathbf{0}\\ \mathbf{0} & \mathbf{S}_2 & \cdots & \mathbf{0}\\ \vdots & \vdots & \ddots & \vdots\\ \mathbf{0} & \mathbf{0} & \cdots & \mathbf{S}_L\end{bmatrix}$$

In the last equality, since the corresponding diagonal blocks are equal, Eq.(19) holds. □

Theorem 4 gives a fast and accurate approach for CoSimRank search on a spanning polytree in a level-by-level style. The CoSimRank scores at level l are immediately computed from those at level $(l-1)$. To retrieve each block $\mathbf{S}_l$ at level l via Eq.(19), it requires only $O(n_l(n_{l-1}+n_l))$ time and $O(n_l^2)$ memory, as opposed to the original method entailing $O(K(m+n)n_l)$ time and $O(m+n)$ memory to retrieve n_l^2 pairs of $\mathbf{S}_l$ scores. Since $n_l \ll n = n_1+n_2+\cdots+n_L$, the complexity improvement of our approach is significant.

EXAMPLE 5. *Consider the spanning polytree T in Figure 2. Theorem 4 computes the CoSimRank $\mathbf{S}(T)$ of T as follows:*

As Level 1 of T has two nodes $\{a,b\}$, Eq.(19) initialises

$$\mathbf{S}_1 = \begin{array}{c|cc} & (a) & (b)\\ \hline (a) & 1 & 0\\ (b) & 0 & 1\end{array}$$

Since $\mathbf{A}_{1,2} = \left[\begin{smallmatrix}.5 & 0\\ .5 & 0\end{smallmatrix}\right]$ and $\mathbf{A}_{2,3} = \left[\begin{smallmatrix}1 & .5 & 1\\ 0 & .5 & 0\end{smallmatrix}\right]$, the CoSimRank similarity of nodes $\{c,d\}$ at Level 2 is computed from $\mathbf{S}_1$:

$$\mathbf{S}_2 = 0.6\mathbf{A}_{1,2}^T\mathbf{S}_1\mathbf{A}_{1,2}+\mathbf{I}_2 = \begin{array}{c|cc} & (c) & (d)\\ \hline (c) & 1.3 & 0\\ (d) & 0 & 1\end{array}$$

Next, the CoSimRank similarity of nodes $\{e,f,g\}$ at Level 3 is computed from $\mathbf{S}_2$:

$$\mathbf{S}_3 = 0.6\mathbf{A}_{2,3}^T\mathbf{S}_2\mathbf{A}_{2,3}+\mathbf{I}_3 = \begin{array}{c|ccc} & (e) & (f) & (g)\\ \hline (e) & 1.78 & 0.39 & 0.78\\ (f) & 0.39 & 1.34 & 0.39\\ (g) & 0.78 & 0.39 & 1.78\end{array}$$

Thus, the CoSimRank scores $\mathbf{S}(T) = diag(\mathbf{S}_1,\mathbf{S}_2,\mathbf{S}_3)$. □

CoSimRank on Static Graph G. After the CoSimRank $\mathbf{S}(T)$ of the polytree T is computed, F-CoSim next computes the CoSimRank changes $\Delta\mathbf{S}$ *w.r.t.* the graph $\Delta G\ (= G\ominus T)$, by utilising our dynamical D-CoSim algorithm in Section 4.1. Finally, the two parts ($\mathbf{S}(T)$ and $\Delta\mathbf{S}$) are added together, which produces the CoSimRank $\mathbf{S}$ of the original graph, *i.e.,*

$$\mathbf{S} = \mathbf{S}(T) + \Delta\mathbf{S}$$

Based on the above ideas, Algorithm 2 provides our complete static scheme, F-CoSim, which incorporates Theorem 4 and our dynamic D-CoSim scheme. F-CoSim consists of three phases: (i) finding a spanning polytree T over G (line 1), (ii) retrieving CoSimRank $\mathbf{S}(T)$ on T (lines 2–7), (iii) computing CoSimRank changes $\Delta\mathbf{S}$ in answer to $(G\ominus T)$ (lines 8–9).

EXAMPLE 6. *Recall G in Figure 2. To retrieve $\mathbf{S}[:,c]$ on G, F-CoSim first decomposes $G = T\oplus(G\ominus T)$. Then, it computes the CoSimRank $\mathbf{S}(T)[:,c]$ of T, as shown in Example 5:*

$$\mathbf{S}(T)[:,c] = \begin{array}{c|ccccccc} & (a) & (b) & (c) & (d) & (e) & (f) & (g)\\ \hline (c) & 0 & 0 & 1.3 & 0 & 0 & 0 & 0\end{array}^T$$

Algorithm 2: F-CoSim (G, C, Q, K)

```
   Input  : graph $G$, decay factor $C$,
            query set $Q$, #-iteration $K$
   Output : CoSimRank scores $\mathbf{S}[:,Q]$ in $G$.
1  set $T :=$ Find_Spanning_PolyTree($G$)
2  set $L_Q :=$ maximum level of the query node of $Q$ in $T$
3  initialise $\mathbf{S}_1 = \mathbf{I}_{n_1}$
4  for $l = 2$ to $L_Q$ do
5      set $n_l :=$ the number of nodes at level $l$ in $T$
6      $\mathbf{A}_{l-1,l}$ is $(n_{l-1}\times n_l)$ col-normalised adjacency block:
       $\mathbf{A}_{l-1,l}[i,j] = 1/\deg_j^-$ if $\exists(i\to j)\in T$; or 0 otherwise
7      compute $\mathbf{S}_l := C\mathbf{A}_{l-1,l}^T\mathbf{S}_{l-1}\mathbf{A}_{l-1,l}+\mathbf{I}_{n_l}$
8  compute $\Delta\mathbf{S}[:,Q] :=$ D-CoSim $(T, G\ominus T, C, Q, K)$
9  compute $\mathbf{S}[:,Q] = diag(\mathbf{S}_1,\mathbf{S}_2,\cdots,\mathbf{S}_L)[:,Q] + \Delta\mathbf{S}[:,Q]$;
10 return $\mathbf{S}[:,Q]$;
```

Next, F-CoSim invokes D-CoSim (Line 8) to compute the CoSimRank increment $\Delta\mathbf{S}[:,c]$ w.r.t. delta graph $(G\ominus T)$:

$$\Delta\mathbf{S}[:,c] = [0,.15,.045,0,.03,.0225,.045]^T.$$

Finally, the CoSimRank score $\mathbf{S}[:,c]$ of G (Line 9) is

$$\mathbf{S}[:,c] = \mathbf{S}(T)[:,c]+\Delta\mathbf{S}[:,c] = [0,.15,1.345,0,.03,.0225,.045]^T$$ □

Correctness. We next show that F-CoSim *correctly* returns the CoSimRank scores $\mathbf{S}[:,Q]$ on G.

THEOREM 5. *Given graph G, the resulting $\mathbf{S}$ returned by Line 10 of F-CoSim is the correct CoSimRank scores over G.*

PROOF. Let $\mathbf{S}(T)$ be the CoSimRank of the polytree T, and $\mathbf{A}(T)$ be the column-normalised adjacency matrix of T. According to Line 1 of F-CoSim, after T is retrieved from G, the column-normalised adjacency matrix $\mathbf{A}$ of G is decomposed into two parts ($\mathbf{A}(T)$ and $\Delta\mathbf{A}$):

$$\mathbf{A} = \mathbf{A}(T) + \Delta\mathbf{A} \quad \text{where} \quad \Delta\mathbf{A} \triangleq \mathbf{A} - \mathbf{A}(T) \tag{20}$$

Theorem 2 guarantees that the CoSimRank of T, denoted as $\mathbf{S}(T)$ ($\triangleq diag(\mathbf{S}_1,\cdots,\mathbf{S}_L)$), that is obtained by Lines 2–7 of F-CoSim, is the correct CoSimRank of T, *i.e.,* $\mathbf{S}(T)$ satisfies the CoSimRank definition:

$$\mathbf{S}(T) = C\mathbf{A}(T)^T\mathbf{S}(T)\mathbf{A}(T) + \mathbf{I} \tag{21}$$

Moreover, our correctness proof of D-CoSim in Theorem 2 guarantees that, by viewing T as the old graph, and $(G\ominus T)$ as the graph update to T, the value of $\Delta\mathbf{S}$, from calling D-CoSim (Line 8 of F-CoSim), is the correct CoSimRank increments *w.r.t.* the update $(G\ominus T)$ to T. That is, $\Delta\mathbf{S}$ satisfies CoSimRank definition:

$$\begin{aligned}\mathbf{S}(T)+\Delta\mathbf{S} = C\cdot(\mathbf{A}(T)+\Delta\mathbf{A})^T\cdot(\mathbf{S}(T)+\Delta\mathbf{S})\cdot\\ \cdot(\mathbf{A}(T)+\Delta\mathbf{A})+\mathbf{I}\end{aligned} \tag{22}$$

According to Line 9, CoSimRank returned by F-CoSim is:

$$\mathbf{S} = diag(\mathbf{S}_1,\cdots,\mathbf{S}_L) + \Delta\mathbf{S} = \mathbf{S}(T) + \Delta\mathbf{S} \tag{23}$$

Plugging Eqs.(21) and (22) into (23) produces

$$\mathbf{S} = C\mathbf{A}^T\mathbf{S}\mathbf{A} + \mathbf{I}.$$

Thus, $\mathbf{S}$ satisfies the CoSimRank definition, *i.e.,* $\mathbf{S}$, returned by F-CoSim, is the correct CoSimRank of G. □

Complexity. Given Q, the time of F-CoSim in each phase is $O(n+m)$, $O(\sum_{l=2}^{L_Q} n_l(n_{l-1}+n_l))$, $O(K(m+np_\ominus|Q|))$, respectively, where L_Q is the maximum level of the query node of Q in T, and $p_\ominus$ is number of update pieces in ΔG. Since $L_Q \le L \ll n$, $n_{l-1}+n_l \ll n$, and $p_\ominus \ll m-n$ in practice, it requires $O(n \max_{2\le l \le L_Q}\{n_{l-1}+n_l\} + K(m+np_\ominus|Q|))$ time in total, as opposed to the $O(Kn(m+|Q|n))$ time of the original method to assess $n \times |Q|$ pairs of scores $\mathbf{S}[:,Q]$.

The memory space of F-CoSim in each phase is $O(m+n)$, $O(m+\sum_{l=1}^{L_Q} {n_l}^2)$, $O(m+Kn)$, respectively. Thus, the total memory is bounded by $O(m+(K+\max_{1\le l\le L_Q}\{n_l\})n)$.

5 EXPERIMENTAL EVALUATION

Our evaluations on various datasets verify the superiority of D-CoSim in dynamic graphs and F-CoSim in static graphs.

The performance efficiency is evaluated by three metrics:
(a) Running Time. On dynamic graphs, D-CoSim quickly answers CoSimRank search. On static graphs, F-CoSim is much faster than the best-known CoSimRank approaches.
(b) Memory Space. Both D-CoSim and F-CoSim require only linear memory, and scale well on million-node graphs.
(c) Accuracy. D-CoSim and F-CoSim do not compromise any accuracy for speedup.

5.1 Experimental Settings

Datasets. We adopt the following public datasets:

Datasets		#-Nodes	#-Edges	Type
as-735	(AS)	7,716	26,467	Undirected
ca-HepPh	(HP)	12,008	237,010	Undirected
email-EuAll	(EE)	265,214	420,045	Directed
web-Google	(WG)	916,428	5,105,039	Directed
wiki-Talk	(WT)	2,394,385	5,021,410	Directed
soc-LiveJournal	(LJ)	18,520,486	298,113,762	Directed

- as-735 (AS). It is a communication graph of autonomous systems, taken from the Border Gateway Protocol logs, where an edge is a who-talks-to-whom relationship.
- ca-HepPh (HP). It is a collaboration graph taken from the arXiv High Energy Physics. If two authors (nodes) co-authored a paper, there is an edge between them.
- email-EuAll (EE). It is an EU email contact graph. Each node is an email address. If node i sent at least one message to j, there is an edge $i \to j$ in the network.
- web-Google (WG). It is a Google web graph, where each node is a web page, and an edge is a hyperlink.
- wiki-Talk (WT). In Wikipedia, each user (node) has a talk page that other users can edit for discussion. In this graph, an edge $i \to j$ means that user i edited user j's talk page.
- soc-LiveJournal (LJ). It is a social community network, where edge $i \to j$ is a friendship link from user i to j.

To simulate real evolution on dynamic graphs, we use RTG (Random Typing Generator) [1] to generate $|\Delta G|$ dynamic updates following linkage generation models [12, 17].

All experiments are conducted on a PC with Intel Core i7-6700 3.40GHz CPU and 64GB memory compiled by VC++.

Compared Algorithms. We implemented our D-CoSim (dynamic) and F-CoSim (static), and compared them with two state-of-art CoSimRank competitors: (a) CSR, a method by [18] that retrieves a CoSimRank score from the sum of the dot product of two Personalised PageRank vectors; (b) CSM, a repeated-squaring method by [27] that cuts down the number of CoSimRank iterations.

Parameters. We chose the following parameters by default: (a) the decay factor $C = 0.8$ and (b) the number of iterations $K = 5$, as previously used in [18].

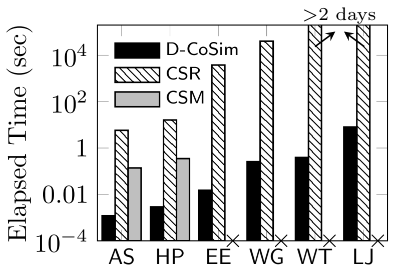
5.2 Experimental Results

5.2.1 Time Efficiency. Figure 3 depicts the time efficiency of D-CoSim on several dynamic graphs. On each dataset, we randomly select $|Q| = 500$ queries, and build $|\Delta G| = 1000$ new edge updates. Figure 3a compares the time of D-CoSim against CSR and CSM to compute CoSimRank changes per update for each query. We see that D-CoSim is consistently 3-5 order-of-magnitude faster than CSR (*resp.*118x faster than CSM). This is because D-CoSim leverages Theorem 1 that evaluates only the refreshed areas of CoSimRank scores in response to graph updates, without the need to recompute all scores from scratch. Moreover, unlike CSM crashes on large datasets (*e.g.,* WT, LJ) due to insufficient memory for repeated squaring memoisation, D-CoSim can update their scores within one second. Figure 3b further depicts the time of D-CoSim *w.r.t.* $|\Delta G|$. As $|\Delta G|$ grows from 500 to 3000 on each dataset, the time of D-CoSim is increasing mildly, highlighting its scalability *w.r.t.* the number of edge updates. It is consistent with the time complexity in Theorem 3 where D-CoSim is linear to the number of update pieces p $(\le \delta)$.

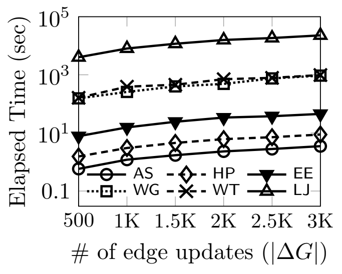
Figure 4 shows the time efficiency of F-CoSim on static graphs. Due to space limitations, we only report the results on three datasets, and the trends on other datasets are similar. Figure 4a compares the time of F-CoSim with CSR and CSM on each dataset. We discern that F-CoSim always outperforms CSR with a speedup up to 9.8x (on EE). Thus, the use of our spanning polytree for fast CoSimRank search is effective (Theorem 4). On HP dataset, CSM is the fastest, but this method only survives on small-scale graphs, due to its high memory storage for repeated squaring. In contrast, F-CoSim scales well on million-edge graphs (*e.g.,* WT, LJ).

Since F-CoSim encompasses three phases (Algorithm 2), Figure 4b details the time allocated in each phase per dataset. We see that, among these phases, Phase 2 (computing $\mathbf{S}(T)$ on spanning polytree T) takes the smallest portion; Phase 1 (finding T from G) the second smallest; Phase 3 (computing $\mathbf{\Delta S}$ *w.r.t.* $G \ominus T$) the largest. This agrees well with our complexity analysis of Algorithm 2, where the time of Phase 2, $O(\sum_{l=2}^{L_Q} n_l(n_{l-1}+n_l))$, is independent of the graph size n, unlike Phases 1 and 3 that hinge on n $(\gg n_l)$.

5.2.2 Memory Efficiency & Scalability. Figure 5 depicts the memory efficiency of D-CoSim and F-CoSim on six real datasets as compared with CSR and CSM. On each dataset, we randomly select $|Q| = 500$ queries. For dynamic graphs,

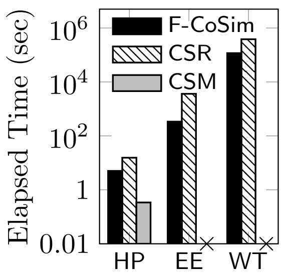


(a) D-CoSim

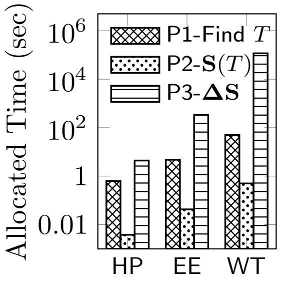


(b) Vary $|\Delta G|$ for D-CoSim

Figure 3: Time Efficiency on Dynamic Graphs

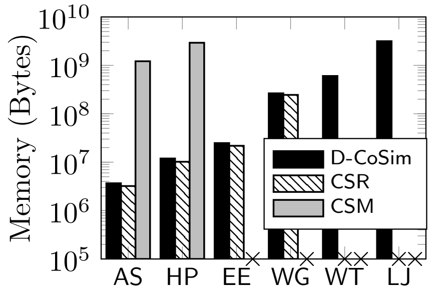


(a) F-CoSim

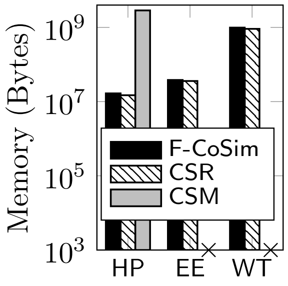


(b) Phases in F-CoSim

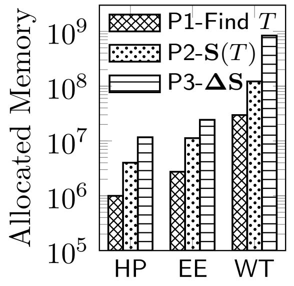
Figure 4: Time Efficiency on Static Graphs



(a) D-CoSim

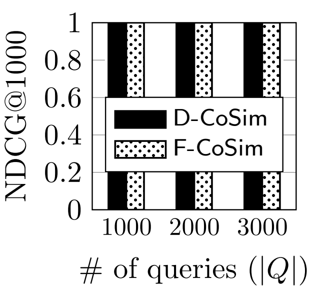


(b) F-CoSim



(c) Phases in F-CoSim

Figure 5: Memory Efficiency & Scalability



(a) Accuracy

Figure 6: Exactness

we generate $|\Delta G| = 1000$ new edge updates to each dataset. Figure 5a reports the memory of D-CoSim for ΔG updates to each dataset *w.r.t.* the query set Q. We see that D-CoSim and CSR have comparable memory; both increase linearly with the growing size of graphs, highlighting the scalability. On small datasets (*e.g.,* AS, HP) when CSM does not fail, the memory of D-CoSim is almost 2.5 orders of magnitude smaller than CSM. This is because D-CoSim requires only linear memory to store auxiliary vectors, as opposed to the $O(n^2)$ memory of CSM for repeated squaring. In Figure 5b, the memory of F-CoSim on static graphs shows the similar tendency. Figure 5c shows the memory usage at each phase of F-CoSim on each dataset. We see that Phase 1 (finding T) has the lowest memory as it is based on linear BFS search. Phase 2 (computing $\mathbf{S}(T)$ on T) requires larger memory than Phase 1, due to its overheads to store the resulting $\mathbf{S}(T)[:,Q]$. These agree with our memory analysis in Algorithm 2.

5.2.3 Accuracy. We evaluate the accuracy of D-CoSim and F-CoSim, relative to the original CSR, on real datasets. We randomly pick up various query sets with its size $|Q|$ varying from 1000 to 3000. For each query set Q, based on the CoSimRank scores $\mathbf{S}[:,Q]$ from D-CoSim (*resp.*F-CoSim), we measure their similarity ranking results via NDCG (Normalised Discounted Cumulative Gain) [24]:

$$\text{NDCG}_Q@k = \tfrac{1}{|Q|}\textstyle\sum_{q=1}^{|Q|}\left(Z_{k,j}\sum_{x=1}^{k}\frac{2^{\mathbf{S}[x,q]}-1}{\log_2(1+x)}\right)$$

where $Z_{k,j}$ is a normalization factor that is the DCG ranking results by the original method CSR. Thus, NDCG = 1 implies that the CoSimRank ranking of the compared algorithm perfectly matches that of CSR, with no accuracy loss. Figure 6 shows the accuracy of D-CoSim and F-CoSim via NDCG for top $k = 1000$ CoSimRank ranking scores on AS. The trends on other datasets are similar. We omit them here. From the results, we notice that, for each query set Q, both NDCGs of D-CoSim and F-CoSim are 1s, implying that D-CoSim and F-CoSim do not sacrifice any accuracy for their speedup. This verifies the correctness of Theorems 2 and 5.

6 CONCLUSIONS

This paper presents a dynamic scheme, D-CoSim, for fast accurate CoSimRank retrieval on evolving graphs. We also apply D-CoSim to static graphs, by proposing F-CoSim to speed up large-scale static CoSimRank retrieval. On dynamic graphs, we devise a novel approach that (a) bunches all edges of ΔG into pieces $\{\Delta G_i\}$ and (b) characterises only the CoSimRank changes in answer to each piece update ΔG_i as the linear combination of vectors, thus discarding unnecessary computations. D-CoSim retrieves CoSimRank quickly and accurately on dynamic graphs, with no need to reassess them from scratch. On static graphs, our fast accurate algorithm, F-CoSim, greatly speeds up CoSimRank retrieval based on three ideas: Given graph G, we (a) find a “spanning polytree” T of G; (b) design an efficient algorithm to retrieve CoSimRank scores $\mathbf{S}(T)$ over T; and (c) apply D-CoSim to evaluate the changes to $\mathbf{S}(T)$ in answer to delta graph ($G \ominus T$). Our empirical studies on various real datasets demonstrate that (a) D-CoSim is 3–5 orders of magnitude faster than the best-known competitors on large dynamic graphs; (b) F-CoSim outperforms the state-of-the-art approaches on static graphs with a speedup up to 9.8 times; (c) D-CoSim and F-CoSim retain comparable linear memory, and scale on million-node graphs, with no compromise of any accuracy for speedup.

REFERENCES

[1] Leman Akoglu and Christos Faloutsos. 2009. RTG: a recursive realistic graph generator using random typing. *Data Min. Knowl. Discov.* 19, 2 (2009), 194–209. https://doi.org/10.1007/s10618-009-0140-7

[2] Ioannis Antonellis, Hector Garcia-Molina, and Chi-Chao Chang. 2008. SimRank++: Query rewriting through link analysis of the click graph. In *WWW*. 1177–1178. https://doi.org/10.1145/1367497.1367714

[3] Bahman Bahmani, Abdur Chowdhury, and Ashish Goel. 2010. Fast Incremental and Personalized PageRank. *PVLDB* 4, 3 (2010).

[4] Orkut Buyukkokten, Hector Garcia-Molina, and Andreas Paepcke. 2001. Seeing the whole in parts: Text summarization for web browsing on handheld devices. In *WWW*. 652–662. https://doi.org/10.1145/371920.372178

[5] Surajit Chaudhuri, Venkatesh Ganti, and Dong Xin. 2009. Exploiting web search to generate synonyms for entities. In *WWW*. 151–160. https://doi.org/10.1145/1526709.1526731

[6] Dániel Fogaras and Balázs Rácz. 2005. Scaling link-based similarity search. In *WWW*. 641–650. https://doi.org/10.1145/1060745.1060839

[7] Yasuhiro Fujiwara, Makoto Nakatsuji, Hiroaki Shiokawa, and Makoto Onizuka. 2013. Efficient search algorithm for SimRank. In *ICDE*. 589–600. https://doi.org/10.1109/ICDE.2013.6544858

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

[9] Minhao Jiang, Ada Wai-Chee Fu, Raymond Chi-Wing Wong, and Ke Wang. 2017. READS: A Random Walk Approach for Efficient and Accurate Dynamic SimRank. *PVLDB* 10, 9 (2017), 937–948. http://www.vldb.org/pvldb/vol10/p937-jiang.pdf

[10] Mitsuru Kusumoto, Takanori Maehara, and Ken-ichi Kawarabayashi. 2014. Scalable similarity search for SimRank. In *SIGMOD*. 325–336. https://doi.org/10.1145/2588555.2610526

[11] Florian Laws, Lukas Michelbacher, Beate Dorow, Christian Scheible, Ulrich Heid, and Hinrich Schütze. 2010. A Linguistically Grounded Graph Model for Bilingual Lexicon Extraction. In *COLING*. 614–622. http://aclweb.org/anthology/C/C10/C10-2070.pdf

[12] Jure Leskovec, Jon M. Kleinberg, and Christos Faloutsos. 2007. Graph evolution: Densification and shrinking diameters. *TKDD* 1, 1 (2007), 2. https://doi.org/10.1145/1217299.1217301

[13] Cuiping Li, Jiawei Han, Guoming He, Xin Jin, Yizhou Sun, Yintao Yu, and Tianyi Wu. 2010. Fast computation of SimRank for static and dynamic information networks. In *EDBT*. 465–476. https://doi.org/10.1145/1739041.1739098

[14] Zhenjiang Lin, Michael R. Lyu, and Irwin King. 2006. PageSim: A novel link-based measure of web page similarity. In *WWW*. 1019–1020. https://doi.org/10.1145/1135777.1135994

[15] Dmitry Lizorkin, Pavel Velikhov, Maxim N. Grinev, and Denis Turdakov. 2010. Accuracy estimate and optimization techniques for SimRank computation. *VLDB J.* 19, 1 (2010), 45–66. https://doi.org/10.1007/s00778-009-0168-8

[16] Phuong Nguyen, Paolo Tomeo, Tommaso Di Noia, and Eugenio Di Sciascio. 2015. An evaluation of SimRank and Personalized PageRank to build a recommender system for the Web of Data. In *WWW*. 1477–1482. https://doi.org/10.1145/2740908.2742141

[17] Alexandros Ntoulas, Junghoo Cho, and Christopher Olston. 2004. What's new on the web? The evolution of the web from a search engine perspective. In *WWW*. 1–12. https://doi.org/10.1145/988672.988674

[18] 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

[19] Tamás Sarlós, András A. Benczúr, Károly Csalogány, Dániel Fogaras, and Balázs Rácz. 2006. To randomize or not to randomize: Space optimal summaries for hyperlink analysis. In *WWW*. 297–306. https://doi.org/10.1145/1135777.1135823

[20] Yingxia Shao, Bin Cui, Lei Chen, Mingming Liu, and Xing Xie. 2015. An Efficient Similarity Search Framework for SimRank over Large Dynamic Graphs. *PVLDB* 8, 8 (2015), 838–849. http://www.vldb.org/pvldb/vol8/p838-shao.pdf

[21] Yizhou Sun, Jiawei Han, Xifeng Yan, Philip S. Yu, and Tianyi Wu. 2011. PathSim: Meta path-based top-K similarity search in heterogeneous information networks. *PVLDB* 4, 11 (2011), 992–1003. http://www.vldb.org/pvldb/vol4/p992-sun.pdf

[22] Akihiro Tamura, Taro Watanabe, and Eiichiro Sumita. 2012. Bilingual Lexicon Extraction from Comparable Corpora Using Label Propagation. In *EMNLP-CoNLL*. 24–36. http://www.aclweb.org/anthology/D12-1003

[23] Boyu Tian and Xiaokui Xiao. 2016. SLING: A Near-Optimal Index Structure for SimRank. In *SIGMOD*. 1859–1874. https://doi.org/10.1145/2882903.2915243

[24] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A Theoretical Analysis of NDCG Type Ranking Measures. In *COLT*. 25–54. http://jmlr.org/proceedings/papers/v30/Wang13.html

[25] Weiren Yu, Xuemin Lin, and Wenjie Zhang. 2014. Fast incremental SimRank on link-evolving graphs. In *ICDE*. 304–315. https://doi.org/10.1109/ICDE.2014.6816660

[26] Weiren Yu, Xuemin Lin, Wenjie Zhang, and Julie A. McCann. 2018. Dynamical SimRank search on time-varying networks. *VLDB J.* 27, 1 (2018), 79–104. https://doi.org/10.1007/s00778-017-0488-z

[27] Weiren Yu and Julie A. McCann. 2015. Co-Simmate: Quick Retrieving All Pairwise Co-Simrank Scores. In *ACL*. 327–333. http://aclweb.org/anthology/P/P15/P15-2054.pdf

[28] 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

[29] Weiren Yu, Wenjie Zhang, Xuemin Lin, Qing Zhang, and Jiajin Le. 2012. A space and time efficient algorithm for SimRank computation. *World Wide Web* 15, 3 (2012), 327–353. https://doi.org/10.1007/s11280-010-0100-6

[30] 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