Unified and Incremental SimRank: Index-free Approximation with Scheduled Principle

Fanwei Zhu[†], Yuan Fang[#], Kai Zhang[†], Yichen Shen[†], Hongtai Cao[⋆], Chengfeng Mao[⋆], Kevin Chen-Chuan Chang[⋆]

[†] Zhejiang University City College, China {zhufanwei@zju.edu.cn}
[#] Singapore Management University, Singapore {yfang@smu.edu.sg}
[⋆] University of Illinois at Urbana-Champaign, USA {hongtai2, cmao kcchang}@illinois.edu

***Abstract*—SimRank is a popular link-based similarity measure on graphs. It enables a variety of applications with different modes of querying (e.g., single-pair, single-source and all-pair modes). In this paper, we propose UISim, a unified and incremental framework for all SimRank modes based on a scheduled approximation principle. UISim processes queries with incremental and prioritized exploration of the entire computation space, and thus allows flexible tradeoff of time and accuracy. On the other hand, it creates and shares common "building blocks" for online computation without relying on indexes, and thus is efficient to handle both static and dynamic graphs. Our experiments on various real-world graphs show that to achieve the same accuracy, UISim runs faster than its respective state-of-the-art baselines in each mode by orders of magnitude, and scales well on larger graphs.**

I. Introduction

Graphs are ubiquitous nowadays, requiring effective similarity measures based on their link structures. Among the link-based similarity measures, SimRank has attracted much attention since it was first proposed by Jeh et al. [8]. The intuition behind SimRank is " two objects are similar if they refer to similar objects," which is recursive with "one object is maximally similar to itself" as the base case. Such intuition naturally simulates human judgements on the similarity of objects based on their connections, and thus has a wide range of applications.

Consider the following scenarios on a DBLP network with interconnected nodes such as researchers, papers and conferences. First, *expert recommendation:* Given a paper p and a researcher r, should r be recommended as an expert to review p? Second, *bibliographic search:* Given a paper p, what are the most relevant papers to p in the entire corpus? Finally, *conference matching:* What are the most similar conference pairs in terms of research topic?

As shown in the above scenarios, there are generally three popular *modes* of the SimRank problem on a graph $G = (V, E)$: *single-pair* SimRank computes the similarity score $s(u, v)$ between a pair of nodes u and v; *single-source* SimRank computes the similarity score between a query node u and every node $v \in V$; *all-pair* SimRank computes the similarity for every pair of nodes in G.

A summary of these SimRank modes is given in Fig. 1, where we formalize a SimRank query as $Q = (A, B)$ with each of A and B being a single node, a subset of nodes, or all the nodes V, and the output $S(Q)$ is the set of corresponding similarity scores.

SimRank Problems		Query $Q = (A, B)$	Output
Popular modes	Single-pair	$A = \{u\}$ $B = \{v\}$	$s(u, v)$: a single SimRank similarity score between u and v
	Single-source	$A = \{u\}$ $B = V$	$[S]_u$: a $\|V\|$-by-1 similarity vector, with each entry $[S]_{u,v} \equiv s(u, v)$
	All-pair	$A = V$ $B = V$	$[S]$: a $\|V\|$-by-$\|V\|$ similarity matrix, with each entry $[S]_{u,v} \equiv s(u, v)$

Fig. 1: SimRank problems on a graph $G = (V, E)$.

A straightforward approach for SimRank is to compute the similarity scores iteratively. Specifically, the SimRank similarity between two nodes u and v is recursively computed based on their in-neighbors $In(u)$ and $In(v)$, as follows [8].

$$s(u, v) = \begin{cases} \dfrac{C}{|In(u)||In(v)|} \displaystyle\sum_{i \in In(u)} \sum_{j \in In(v)} s(i, j) & u \neq v \\ 1 & u = v \end{cases} \quad (1)$$

However, the computation is expensive even on a moderately large graph due to its iterative nature. Thus, many works have devoted to speedup SimRank computation with approximation. We summarize three major challenges in SimRank approximation and motivate our solution as follows– the detailed study of existing works can be found in Sect. II.

First, as there are distinct modes of SimRank for different scenarios, it is desirable to support all different modes in a unified manner by one algorithm for simplicity and robustness of system maintenance. In contrast, virtually all existing algorithms are designed for specific modes– *E.g.*, ProbeSim [16], the state-of-the-art method based on Monte Carlo simulation, samples random tours from a "single-source" query node which cannot be naturally extended to sampling for single-pair queries where the two ends are fixed and must meet.

Second, as different applications may have specific requirement of the approximation, *e.g.*, some online tasks emphasize on a fast estimate while some others may rely on more accurate scores, it is desirable to support flexible tradeoffs of efficiency and accuracy. In contrast, most other algorithms exhibit often a narrow range of tradeoff– *E.g.*, ProbeSim's random trials

requires a certain amount of minimal "significant" samples of the computation space, which restricts its range of tradeoffs.

Third, as most real-world graphs are dynamic with frequent updates (*e.g.*, social networks such as Twitter), it is desirable to support efficient online computation without relying indexes. In contrast, many other algorithms need to precompute and maintain an index to process online queries, and thus are not flexible to handle dynamic graphs–*E.g.*, FLPMC, FBLPMC [24], the state-of-the-art index-based single-pair SimRank algorithms needs 100ms to 1s to update its index for each edge insertion or deletion on medium-sized graphs, and with the increasing of graph size, the index update time grows exponentially.

Our principle. Motivated by the three challenges, we propose a unified and incremental framework, UISim, to efficiently process different modes of SimRank queries with a *scheduled approximation* principle. Our approach is built on the *random surfer-pair model* [8] where the SimRank similarity $s(u, v)$ is conceptually interpreted as the probability that two random surfers can meet if they randomly walk *backwards* on the graph G, from nodes u and v respectively.

Specifically, UISim has three major ingredients–*unified computation space*, *prioritized exploration of the space*, and *online sharing of common computation*– which are expected to tackle the above challenges.

First, *to support unification of different modes*, it identifies a "computation space of query tours" that naturally adaptable to each distinct mode– For any SimRank query $Q = (A, B)$ where each of A and B is set of query nodes, its computation space is conceptually viewed as the aggregate of necessary random walk tours starting from A and B. Thus, to calculate any similarity scores $S(Q)$, we can simply enumerate the set of corresponding query tours T_Q and process them in a unified framework– all tours in T_Q aggregate to the exact scores, while a subset of tours gives an approximation.

Second, *to support flexible tradeoff of time and accuracy*, it suggests "a prioritized exploration of the computation space" to gradually cover the query tours in an important-first manner– T_Q is further partitioned into disjoint subsets $T_Q = T_Q^0 \cup \cdots \cup T_Q^\eta$ such that tours in any T_Q^i are more important than tours in T_Q^{i+1}. We then handle T_Q through multiple iterations, with each iteration i computing a SimRank *increment* $\hat{S}^i(Q)$ over the tours in T_Q^i, adding up to an overall estimate $\hat{S}^{(\eta)}(Q) = \hat{S}^0(Q) + \cdots + \hat{S}^\eta(Q)$ after η iterations. Unlike random sampling, our incremental exploration is deterministic, intentionally prioritized, and thus we can support a wide range of tradeoffs without being burdened by statistically-necessary minimal sampling.

Third, *for efficient computation without relying indexes*, it allows us to create and share common "building blocks" computed on-the-fly to accelerate the iterative computation– We factorize the query tours into fine-grained segments that shared across iterations, and organize them to create basic computation units which can be easily computed and reused online. Thus, each SimRank increment $\hat{S}^i(Q)$ can be efficiently derived from the "assembling" of common building blocks. Unlike other indexed approaches, our principle achieves high efficiency by sharing online computations rather than relying on precomputed indexes, and thus works well on both static and dynamic graphs.

Concrete realization. Although the principle is promising to achieve our goals, to concretize it in SimRank setting is challenging due to the *complex* query tours and the *diverse* query modes:

- *First, complex query tours.* The query tours T_Q SimRank deals with are complex two-side "first-meeting" tours [14] starting from A and B on each side, rather than the regular tours from one node to another. Thus, to explore the computation space, we first need to construct $T_Q = T_Q^0 \cup \cdots \cup T_Q^\eta$ from regular one-side tours on G in a prioritized manner for flexible tradeoff of time and accuracy.
- *Second, diverse query modes.* SimRank has a variety of concrete modes, each of which has its own requirement to identify the computation space. Thus, we need to efficiently *specialize* the generation of each T_Q^i for different mode of queries such that a complex query (*e.g.*, single-source SimRank $s(u)$) can be better processed than trivially repeating a set of the basic queries (*e.g.*, single-pair SimRank $s(u, v)$ for each $v \in V$).

To concretely realize the principle, in Sect. III we investigate the necessary query tours in different SimRank modes, and propose to unify their computation space with the assembling of two query-specific "partial-tour" sets $P_A \bowtie P_B$. We then develop a hub-based benefit model to partition those partial tours and assemble their partitions in an incremental and prioritized manner such that the query tours that bring more contribution in the computation would be generated earlier in Sect. IV. We further identify the shared tour segments in different partial-tour partitions and propose a subgraph expansion model to use those substructure as building blocks to speed up the iterative online computation in Sect. V. We analyze the complexity and error bound of UISim in Sect. VI.

Empirical evaluation. We conduct extensive experiments on various real-world datasets in Sect. VII. We first compare UISim with the state-of-the-art baseline in single-pair and single-source modes, and find out UISim significantly outperforms its respective baselines in each mode– UISim is up to 1- 2 orders of magnitude faster on smaller graphs, and up to 4 orders of magnitude faster on large graphs, *i.e.*, compared to the strongest baselines designed specifically for each mode, to achieve the same level of accuracy, the running time of the unified UISim is significantly less than that of the baseline. We also validate the scalability of UISim in growing graphs.

II. Related Work

Numerous studies have been devoted to speeding up the computation of SimRank on a single machine, which fall into three main categories in the following. Note that, there are also a few methods focusing on parallelizing SimRank computation [15], [7], [27] or investigating SimRank problems

on uncertain graphs [33], [3], which are orthogonal to our work and excluded in the comparison.

Iterative methods. Many approaches [17], [18], [28], [14] directly optimize the basic iterative algorithm, by reducing unnecessary computation and reusing shared computation both within and across iterations. Lizorkin et al. [17], [18] proposes to memorize the reusable partial sums across iterations to prevent repeated computation for all-pair SimRank. Yu et al. [28] further reduces the redundancy in computing partial sums with sub-summation sharing in all-pair mode. Li et al. [14] employs position probability to reduce the computation not relevant to a query in the single-pair mode. Approximations are often used in the above methods, which typically ignore intermediate similarities below a small threshold to further speed up the computation. However, even the state-of-the-art iterative methods [28], [14] require $O(k|V|^2)$ time for k iterations in the worst case, which is still infeasible to handle large graphs.

Linear system solution. Another line of research transforms the iterative SimRank equation into linear system representation, and applies the linear algebra techniques such as matrix decomposition to approximation SimRank. Li et al. [13] derives a linear system and performs singular value decomposition (SVD) on the similarity matrix to get SimRank approximation. Fujiwara et al. [6] proposes SimMat that computes SimRank based on the Sylvester equation and low-rank approximation of the similarity matrix. Yu et al. [29] relaxes the constraint that the graph should be non-singular and provides a treatment of SimMat, by supporting similarity assessment on non-invertible adjacency matrices. Wang et al. [25] proposes a new closed-form solution of exact SimRank matrix, based on which a local push algorithm is developed for all-pairs SimRank computation. The linear system based methods breaks the holistic nature of SimRank computation, however, they cannot guarantee the first-meeting constraint in the original SimRank definition. Moreover, they require quadratic time to obtain a low-rank representation and loss accuracy from the optimization techniques. *Random walk-based approximation.* To handle large graphs, the majority of studies solve SimRank based on random walks. Fogaras et al. [5] applies MC simulation to sample random walk paths between two nodes, which addresses single-pair SimRank. Kusumoto et al. [10] later extends it to address the single-source mode through extensive pruning. Maehara et al. [19], [20] proposes to use MC simulation to estimate a diagonal correction matrix which will be applied to answering SimRank queries. Although MC methods are promising in handling large graphs, they can only achieve a higher level of accuracy through more and more samples at the cost of efficiency. Recently, Wang et al. [24] propose to combine the local push technique[25] with MC sampling to reduce the sample size. However, the worst case complexity of the proposed index-free version BLPMC is the same as the pure MC sampling, while a more efficient version FLPMC relies on an index precomputed on a conceptual graph with $|V|^2$ nodes.

Comparison to our work. First, in terms of problem, UISim proposes to unify all three modes of SimRank with the scheduled approximation principle, and develops mode-specific techniques to efficiently handle different SimRank queries. On the contrary, most previous work only focuses on one specific mode of SimRank problem (*e.g.*, references [14], [5] for single-pair, [11] for single-source, and [18], [28], [26], [23] for all-pair). Although some recent work [30], [19] also address different modes of SimRank, their techniques are essentially designed for certain modes. In particular, Yu et al. [30] develops an optimized technique for single-source SimRank only, and proposes to decompose the partial-pair problem (*i.e.*, SimRank similarity between any two sets of nodes) into multiple single-source problems. Maehara et al. [19] proposes linearized technique to efficiently tackle single-pair and single-source, while the all-pair problem is solved by trivially repeating the single-source solution. Jiang et al. [9] discusses the algorithms for single-pair and single-source, where single-source takes $O(|V|r)$ query time, $|V|$ times more than single-pair.

Second, in terms of technique, our work follows the line of approximating SimRank over random walk tours. However, instead of randomly stimulating fingerprints, we structurally organize all the tours in the computation space based on their importance, and enumerate them in a prioritized way. Thus, UISim has two distinct properties, "important-first" and "incrementally-enhanced", compared to existing works in the same line. Note that, a similar scheduled principle is applied to estimating Personalized PageRank values in FastPPV [31], [32], but their problem and solution are different from SimRank– we face unique challenges in SimRank setting and requires novel realization techniques as we present in Sect. I. The most relevant work to UISim is TopSim-based algorithms [11] which are also based on path enumeration rather than random stimulation. However, TopSim-based algorithms only consider random walk tours in the neighborhood of query node. Thus, they are quite sensitive to the graph structure-both accuracy and efficiency are affected by the density of graphs, *i.e.*, for different graphs, the number or importance of enumerated tours cannot be guaranteed. On the contrary, UISim allows a wide range of tradeoff of time and accuracy by gradually cover the tours in the entire computation space.

Third, in terms of applications, UISim is capable of efficiently handling both static and dynamic graphs. Different from the index-based algorithms[21], [24] which needs expensive cost to update their index on dynamic graphs, UISim runs all the computations at query time and thus can support real-time queries on any graphs. There are also some index-free algorithms[16], [11] proposed to support dynamic updates, but UISim outperforms them in the query efficiency as we factorize the tours handled in iteration into fine-grained building blocks that can be computed efficiently online and shared across iterations. Moreover, as UISim allows flexible tradeoff of time and accuracy, it supports a wide range of applications with different requirement of the approximation.

Notation	Description
$Q = (A, B)$	a SimRank query defined on query nodes A and B.
T_Q, T_Q^i	the set of query tours *w.r.t.* Q; the i-th important partition of T_Q.
$s(u, v)$	the exact SimRank similarity between node u and v.
$\hat{s}(u, v)$	the estimated SimRank similarity between u and v.
$\hat{s}^i(u, v)$	the i-th SimRank increment computed over T_Q^i.
P_u, P_u^i	the set of partial tours ending at u; the i-th important partition of P_u.
$P_u \otimes P_v$	the constrained assembling of P_u and P_v.
$\mathcal{L}(t)$, $\mathcal{L}_h(t)$	the natural length of tour t; the hub length of t.
$G(u\|v)$	the set of partial tours from v to u, $G(u\|*)$ denotes the in-subgraph of u and $G(*\|v)$ denotes the out-subgraph of v.
$G^i(u\|v)$	the set of hub-length-i tours from v to u.
$G^{i,l}(u\|v)$	length-l tours in $G^i(u\|v)$.
H_u^i	the border hubs of u, *i.e.*, hubs on the border of $G^i(u\|*)$ or $G^i(*\|u)$.
$r^i(u\|v)$	the reachability of tours in $G^i(u\|v)$.

Fig. 2: Main notations.

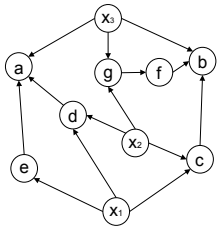

Fig. 3: A toy graph G.

Query Tours	SimRank Modes		
$t_1: a \leftarrow x_3 \rightarrow b$	Single pair $Q = (a, b)$	Single source $Q = (a, V)$	All Pair $Q = (V, V)$
$t_2: a \leftarrow e \leftarrow x_1 \rightarrow c \rightarrow b$			
$t_3: a \leftarrow d \leftarrow x_2 \rightarrow c \rightarrow b$			
$t_4: a \leftarrow d \leftarrow x_1 \rightarrow c \rightarrow b$			
$t_5: a \leftarrow x_3 \rightarrow g$			
$t_6: a \leftarrow d \leftarrow x_2 \rightarrow g \rightarrow f$			
$t_7: d \leftarrow x_2 \rightarrow g$			
..... (the others are omitted)			
$t_{14}: f \leftarrow g \leftarrow x_2 \rightarrow c \rightarrow b$			

Fig. 4: Query tours in different modes.

III. Unified Computation space: aggregating tours

As motivated in Sect. I, there are distinct modes of SimRank in real applications, requiring a unified algorithm to process different queries. To support unified SimRank, we first investigate the computation space of three popular SimRank modes: *single-pair*, *single-source* and *all-pair* SimRank.

To illustrate, we introduce a toy graph G (Fig. 3) and list the example tours in different modes (Fig. 4). We observe that for any SimRank query $Q = (A, B)$ (*e.g.*, $A = \{a\}$ and $B = \{b\}$), the computation space is composed of a set of two-side tours ended with nodes in A (*e.g.*, node a) and B (*e.g.*, node b) on each side and centered at a set of meeting nodes in random walks.

Conceptual view of computation space. Conceptually, we can model the computation space of any SimRank query $Q = (A, B)$ as a set of *query tours* T_Q *w.r.t.* the query nodes A and B, formalized as:

$$T_Q = \{u \leftarrow u_1 \leftsquigarrow u_k \leftarrow x \rightarrow v_k \rightsquigarrow v_1 \rightarrow v \mid u \in A, v \in B, x \in X, \forall i \in [1, k], u_i \neq v_i\} \quad (2)$$

where $X = \{x \mid x \in V, |Out(x)| \geq 2\}$ is defined as the set of meeting nodes, *i.e.*, any node x with at least two out-neighbors $|Out(x)| \geq 2$ on G.

Then any SimRank similarity can be interpreted as the *first-meeting probability* of two backward random surfer on G, starting from A and B respectively,*i.e.*, the aggregated reachabilities of tours in T_Q. Specifically, $s(u, v)$ can be calculated by aggregating the reachabilities of the two-side *first-meeting* tours, $u \leftsquigarrow x \rightsquigarrow v$, which end with u and v on each side [14]:

$$s(u, v) = \sum_{t \in \{u \leftsquigarrow x \rightsquigarrow v\}} R(t) \quad (3)$$

Note that, we use $\leftsquigarrow$ ($\rightsquigarrow$) to denote a sequence of edges and $\leftarrow$ ($\rightarrow$) to denote a direct edge throughout this paper.

Concrete tour space. To partition T_Q for a scheduled approximation, we notice that T_Q is a set of complex "two-side" tours which can not be directly identified on G–they have to be assembled (or partialized) first. Specifically, each *two-side* query tour should be assembled from two *regular* tours on G. For example, for a single-pair query $Q = (a, b)$, the query tours $T_{(a,b)}$ is built by assembling the same-length regular tours from the same meeting nodes to a and b respectively. Formally, for any two regular tours $p_u : x \rightsquigarrow u$ and $p_v : x' \rightsquigarrow v$ on G, we define $p_u \circ p_v$ as the *constrained assembling* of p_u and p_v if they 1) start at the same meeting node, and 2) have the same length. That is:

$$p_u \circ p_v \equiv u \leftsquigarrow x \rightsquigarrow v \quad \textit{iff} \quad x = x' \text{ and } \mathcal{L}(p_u) = \mathcal{L}(p_v) \quad (4)$$

where $\mathcal{L}(p)$ denotes the length of an arbitrary tour p. To avoid ambiguity, we also refer a two-side query tour as a *full tour*, and each regular tour as a *partial tour*.

Generally, the query tours T_Q in any SimRank modes can be assembled from two corresponding partial-tour sets P_A and P_B. Let P_U denote the set of partial tours ending at a node $u \in U$, *i.e.*, $P_U = \{p : v \rightsquigarrow u \mid u \in U, v \in V\}$, in realization, we construct T_Q as:

$$T_Q \equiv P_A \bowtie P_B = \{p_a \circ p_b \mid p_a \in P_A; p_b \in P_B\} \quad (5)$$

Note that, in Eq. 4, we relax the *first-meeting constraint*, *i.e.*, two partial tours should meet at **only one** node, to simplify the discussion. We will discuss how to correct the reachability of *multi-meeting* tours, *i.e.*, full tours that have more than one meeting node in Appendix. Several previous works [21], [23] also relax the constraint for efficient computation, but they use different correction strategies tailored to their algorithms.

Such unification of computation space is naturally adaptable to each distinct SimRank mode:

- First, any SimRank query $S(Q)$ can be processed by incrementally aggregating the reachabilities of certain query tours T_Q– all tours in T_Q aggregate to the exact scores, while tours in certain partitions T_Q^i give an approximation.
- Second, the scheduled approximation principle (see Sect. I) applies to any mode by enumerating and prioritizing the corresponding partial tours P_A and P_B, which we will discuss in Sect. IV.

IV. Incremental approximation: realizing with partial tours

We now discuss how to concretely realize the scheduled approximation principle with partial tours to support flexible tradeoff of time and accuracy. Specifically, to incrementally generate the partitions of T_Q, we will explore how to partition partial tours (*e.g.*, P_A) into subsets (*e.g.*, $P_A = P_A^0 \cup \cdots \cup P_A^i$), and schedule the assembling of these partitions (*e.g.*, $P_A^i \bowtie P_B^j$) in a way that the full tours generated earlier would bring more accuracy improvement to the computation.

Conceptually, we define the *benefit* of a partial-tour assembling as the accuracy improvement from handling the assembled tours, and propose to schedule the assembling of partial tours based on their benefit. As the benefit of an assembling depends on the importance of each full (assembled) tour and the number of full tours, *i.e.*, handling more important tours would better improve the accuracy of estimation, we develop two rules to schedule the assembling of partial tours as follows:

- *Rule 1 (Important-First): Important partial tours assembled earlier.* As the reachability (*i.e.*, importance) of any full tour $R(p_u \circ p_v)$ can be computed as the product of its partial tours' reachability $\frac{R(p_u)R(p_v)}{C^{\mathcal{L}(p_u)}}$, by assembling the important partial tours earlier, we can also obtain the important full tours earlier.
- *Rule 2 (Symmetric-Preferred): Symmetric partitions of partial tours assembled earlier.* As each full tour must be symmetric (in terms of tour length) at the meeting node, by assembling symmetric partial tours, we can expect more valid matches.

Guided by the two rules, we now propose a *hub-based benefit model* to concretely partition the partial tours and incrementally assemble their partitions.

First, partitioning partial tours. To partition partial tours, we need a simple yet effective metric to quantify the above rules. In the SimRank setting, the reachability of a specific partial tour $p : x \to w_1 \to \cdots \to w_k$ with length $\mathcal{L}(p)$, is the probability of reaching x from w_k through p in a random walk where at each step, the random surfer would go to one of its in-neighbors, with probability C, *i.e.*, the damping factor in random walks. That is,

$$R(p) \triangleq C^{\mathcal{L}(p)} \prod_{i=1}^{\mathcal{L}(p)} \frac{1}{|In(w_i)|} \tag{6}$$

Therefore, the importance of partial tours can be measured by the number of *hub nodes*, *i.e.*, high-degree nodes that significantly decay the reachability of the tour, it passes through. On the other hand, the symmetry of two partial tours can also be determined by the number of high-degree nodes they pass through, as the tours passing through the same number of high-degree nodes tend to be symmetric in terms of their natural length. In summary, the number of high-degree nodes is effective to measure both tour importance and symmetry. We also refer to the number of hub nodes in a tour p (excluding the starting node as it does not decay the reachability of p) as the *hub length* of p, denoted by $\mathcal{L}_h(p)$.

Therefore, given a set of hub nodes H selected on G, any partial tour set P_u can be partitioned into η disjoint subsets P_u^i, each of which contains only the tours of hub length i, formalized as:

$$P_u = P_u^0 \cup \ldots \cup P_u^\eta \text{ s.t. } \forall i \in \{0, \ldots \eta\}, P_u^i = \{p \mid p \in P_u, \mathcal{L}_h(p) = i\} \tag{7}$$

Second, assembling full tour. With the hub length notion, we can concretize the two rules to prioritize the assembling of partial-tour partitions: 1) According to the *Important-First* Rule, any two partitions with a smaller sum of hub length should be assembled earlier; 2) According to *Symmetric-Preferred* Rule, partitions with a smaller difference in the hub length should be assembled earlier.

More formally, for any two assemblies $A_{ij} : P_u^i \bowtie P_v^j$ and $A_{i'j'} : P_u^{i'} \bowtie P_v^{j'}$, we should schedule A_{ij} in an earlier iteration than $A_{i'j'}$, denoted by $A_{ij} \prec A_{i'j'}$, with the following criteria:

$$A_{ij} \prec A_{i'j'} \text{ if } \begin{cases} i + j \le i' + j' \\ \text{and} \\ |i - j| \le |i' - j'| \end{cases} \tag{8}$$

We also notice that, the two rules may conflict sometimes. For example, consider two assemblies $P_u^0 \bowtie P_v^1$ with $P_u^1 \bowtie P_v^1$, the first one should be scheduled earlier according to Rule 1, while it should be scheduled later according to Rule 2. Generally, when the order of two assemblies conflicts by each individual rule, the benefit of their assembling can not be differentiated, and thus can be scheduled in either order.

We integrate the two rules into $Max(i, j)$ and use it as the overall priority index of any $P_u^i \bowtie P_v^j$, since $Max(i, j)$ equals $(i+j)+|i-j|$, *i.e.*, larger $i+j$ and larger $|i-j|$ would result in a larger $Max(i, j)$, and thus has a higher priority to be scheduled. Note that, other metrics are also possible as long as they are consistent with Eq. 8, and easy to check.

Now, we are able to generate the full tours through iterations to incrementally evaluate SimRank. For any two partitions P_u^i and P_v^j, they will be assembled in iteration $Max(i, j)$. In other words, in iteration k, all the partial-tours assemblies $P_u^i \bowtie P_v^j$ with $Max(i, j) = k$ would be scheduled to generate a set of full tours, formalized as:

$$T_{(u,v)}^k = \bigcup_{Max(i,j)=k} P_u^i \bowtie P_v^j \tag{9}$$

Thus, the k-th SimRank increment $\hat{s}^k(u, v)$ is calculated as:

$$\hat{s}^k(u, v) = \sum_{Max(i,j)=k} R(P_u^i \bowtie P_v^j) \tag{10}$$

and the SimRank score $\hat{s}^{(\eta)}(u, v)$ estimated after iteration-η is:

$$\hat{s}^{(\eta)}(u, v) = \sum_{k=0}^{\eta} \sum_{Max(i,j)=k} R(P_u^i \bowtie P_v^j) = \sum_{i,j \le \eta} R(P_u^i \bowtie P_v^j) \tag{11}$$

The incremental approximation with prioritized assembling of partial tours allows flexible tradeoff of accuracy and time– a fast yet good estimate can be obtained with a small η, which can be further enhanced by increasing the number of iterations.

V. Index-free solution: sharing across iterations

To process SimRank queries in the incremental manner (Eq. 11) without relying on any precomputed indexes, we now further examine the specific query tours in each iterations for efficient online realization. Specifically, given any query $Q = (A, B)$, we will investigate how to efficiently span the partial tours P_u^i, P_v^j and generate the valid full tours.

First, *partial tours spanning.* To motivate, let's examine the partial tours in the first two iterations of estimating $s(a, b)$ (*i.e.*, $\hat{s}^0(a, b)$ and $\hat{s}^1(a, b)$) in our toy graph.

We observe that the all the tours P_a^1 (*e.g.*, $a \leftarrow d \leftarrow x_1$) in iteration-1 can be "extended" from the tours ended with hubs in P_a^0 (*e.g.*, $a \leftarrow d$), by adding corresponding "extension" tours at that hub node (*e.g.*, $d \leftarrow x_1$). The reason behind such extension is because we partition partial tours by their hub length– tours in P_u^i are one hub-length shorter than tours in P_u^{i+1} and thus can be viewed as the "prefix" tours of P_u^{i+1}. Generally, we use a *graph expansion model* to illustrate such tour extension. We refer to the set of any partial tours P_u^i (*i.e.*, hub-length-i tours ending at u) as the *i-level in-subgraph* of u, as they actually form a subgraph of incoming tours to u, formalized as $G^i(u|*) = \{p : w \nleftarrow v \mid v \in V; \mathcal{L}_h(p) \leq i\}$. The 0-level in-subgraphs are also referred to as the prime in-subgraphs. Then, by expanding the *(i-1)-level in-subgraph* of u at its "border" hubs, denoted by H_u^{i-1}, with the hub-length-0 "extension" tours ending at hub, we can obtain the i-level in-subgraph $G^i(u|*)$ consisting of the hub-length-i partial tours:

$$G^i(u|v) = \bigcup_{h \in H_u^{i-1}} G^{i-1}(u|h) \oplus G^0(h|v) \tag{12}$$

The reachability of the hub-length-i partial tours can be "extended" similarly. Formally, let $r^i(u|v)$ be the overall reachability of the extended tours in $G^i(u|v)$, we have:

$$r^i(u|v) = \sum_{h \in H_u^{i-1}} r^{i-1}(u|h) \cdot r^0(h|v) \tag{13}$$

Such hub-by-hub graph expansion allows us to efficiently enumerate the partial tours in each iteration by dynamically creating and sharing the common "building blocks" across iterations, *i.e.*, the prime subgraphs of hub nodes– on the one hand, once a prime subgraph is computed, it can be reused in later iterations to build longer tours, as the set of partial tours in any iteration are assembled from *hub segments* (*i.e.*, hub-length-0 tours); on the other hand, the prime subgraphs can be efficiently computed on-the-fly as it only consists of the hub-length-0 tours in the neighborhood of certain nodes

Next, to *assemble full tours* over partial-tour partitions, we notice that partial tours in a partition can have different natural length, while the valid query tours should have the same length on either side of the meeting nodes by definition (Eq. 4). Thus, to assemble two partial-tour sets, we can skip those "mis-matching" partial tours as they are not able to generate valid full tours. Accordingly, the aggregated reachability of full tours in $G^\eta(u|*) \bowtie G^\eta(v|*)$, can be obtained by assembling the reachability of length-matched partial tours that start at the same meeting node.

$$R\left(G^i(u|*) \bowtie G^j(v|*)\right) = \sum_{x \in X} \sum_{l \leq M} \frac{1}{C^l} \left(r^{i,l}(u|x) \cdot r^{i,l}(v|x)\right) \tag{14}$$

where M is the maximal natural length in computation (*i.e.*, the number of iterations required for the fixed-point method to converge [8]), and in $r^{i,l}(u|x)$ we expand the superscript of hub length i to also denote natural length l as i, l.

Specification for other modes. Such the *hub-by-hub* extension of partial tours and *length-by-length* matching of full tours can naturally apply to different SimRank modes as we explained in the unified principle. But since the partial tours P_A and P_B in different modes can have different forms, *i.e.*, they can be ending at a single node, a subset of V, or any node in V, we can utilize the special properties of partial tours in each mode to design more efficient implementations.

We start with two single nodes. To span the partial tours from u to v (*i.e.*, $\{u \rightsquigarrow v\}$), we can enumerate the *incoming* tours of v from u, or *outgoing* tours of u to v. Such enumeration can be done by growing a *subgraph* from v or u at different directions– expanding the *in-subgraph* of v, denoted by $G(v|*) = \{p : v \nleftarrow w \mid w \in V\}$, or *out-subgraph* of u, denoted by $G(*|u) = \{p : u \rightsquigarrow w \mid w \in V\}$.

Now consider how to efficiently span a set of partial tours P_U. Since to assemble full tours, the valid partial tours in P_U should start from certain meeting nodes, we are able to compare the number of query nodes $|U|$ with the number of reachable meeting nodes $|X_U|$ to decide the directions of subgraph expansion. If U only consists of a single node u (as in the single-pair mode), we should expand an in-subgraph $G(u|*)$ to obtain P_U, since the other way of expanding an out-subgraph $G(*|x)$ for *each* meeting node x would waste more effort in tours that do not end at u. *I.e.*, we choose to expand from U since $|U| = 1 \ll |X|$. In contrast, if tours in P_U are from X to V, *i.e.*, $U = V$, we would instead expand out-subgraphs $G(*|x)$ from each meeting node x. That is, we now choose to expand from X, since $|X| < |U| = |V|$. Generally, we should expand the set with fewer nodes– expanding $|P_U|$ in-subgraphs from query nodes U if $|P_U| < |X_U|$ or $|X_U|$ out-subgraphs from meeting nodes X_U if $X_U < |P_U|$.

Therefore, for *single-pair SimRank*, we expand the in-subgraphs of query nodes u and v, *i.e.*, $G(u|*)$ and $G(v|*)$, and assemble them at the common meeting nodes $\bigcup_{x \in X} G(u|x) \bowtie G(v|x)$. For *single-source SimRank*, we first expand $G(u|*)$ to span P_u, and then expand $G(*|x)$ for each meeting node $x \in X_u$ to assemble the full tours $\bigcup_{x \in X_u} G(u|x) \bowtie G(*|x)$. For all-pairs SimRank, as generally $|X| << |V|$, we expand the out-subgraph of each $x \in X$ and assemble full tours $\bigcup_{x \in X} G(*|x) \bowtie G(*|x)$.

Details of the unified index-free solution for three SimRank modes are illustrated in Algorithm 1. First, we select a set of hub nodes H on the input graph G– Given $|H|$, the number of hubs, the $|H|$ nodes with the highest in-degree are chosen as hubs (Line 1). In our current discussion, we only explore the decaying power of hubs for discriminating tours, and thus we select hub nodes by their in-degree (*i.e.*, higher in-degree indicates higher decaying power). The number of hubs depends on the structure of graph, which we will explain in the Sect. VII. Then we chose mode-specific graph expansion technique to compute the reachability of partial tours, which will be further assembled in a length-aware manner at the same meeting nodes and aggregate to the overall approximation. The subroutine of incremental graph expansion is sketched in Algorithm 2.

Algorithm 1: Incremental and Unified SimRank approximation

Input: a graph G; number of hub nodes H; number of iteration η; query $Q = (A, B)$; max tour length M
Output: estimated SimRank $\hat{S}^{(\eta)}(Q)$

$H \leftarrow$ Select $|H|$ hubs on G;
if $A = \{u\}, B = \{v\}$ **then** /* Single-pair */
  $r^{(\eta)}(u|*) \leftarrow GraphExp(G, \eta)$;
  $X_u \leftarrow$ meeting nodes in $r^{(\eta)}(u|*)$;
  $r^{(\eta)}(v|*) \leftarrow GraphExp(G, \eta)$;
  $X_v \leftarrow$ meeting nodes in $r^{(\eta)}(v|*)$;
  foreach $x \in X_u \cup X_v$ **do**
    foreach $l \in [1, M]$ **do**
      $s(u, v) \leftarrow s(u, v) + r^{(\eta),l}(u|*) r^{(\eta),l}(v|*)$;
      $\hat{S}^{(\eta)}(Q) \leftarrow s(u, v)$
    end
  end
if $A = \{u\}, B = V$ **then** /* Single-source */
  $r^{(\eta)}(u|*) \leftarrow GraphExp(G, \eta)$;
  $X_u \leftarrow$ meeting nodes in $r^{(\eta)}(u|*)$;
  foreach $x \in X$ **do**
    $r^{(\eta)}(*|x) \leftarrow GraphExp(G, \eta)$;
    foreach $v \in V$ **do**
      foreach $l \leq M$ **do**
        $s(u, v) \leftarrow s(u, v) + r^{(\eta),l}(u|*) r^{(\eta),l}(*|x)$;
        $[\hat{\mathbf{S}}]_{u,v} \leftarrow s(u, v)$;
      end
    end
  end
  $\hat{S}^{(\eta)}(Q) \leftarrow [\hat{\mathbf{S}}]$;
if $A = B = V$ **then** /* All-pair */
  $X \leftarrow$ *meeting nodes in* G;
  foreach $x \in X$ **do**
    $r^{(\eta)}(*|x) \leftarrow GraphExp(G, \eta)$;
    foreach $v \in V$ **do**
      foreach $l \leq M$ **do**
        $s(u, v) \leftarrow s(u, v) + r^{(\eta),l}(u|x) r^{(\eta),l}(v|x)$;
        $[\hat{\mathbf{S}}]_{u,v} \leftarrow s(u, v)$;
      end
    end
  end
  $\hat{S}^{(\eta)}(Q) \leftarrow [\hat{\mathbf{S}}]$;
return $\hat{S}^{(\eta)}(Q)$.

Algorithm 2: GraphExp (Subroutine)

Input: a graph G; a root node u; type of subgraph κ, number of iterations η
Output: reachability over η-level subgraph $r_u^{(\eta)}$

if $\kappa =$ 'I' **then**
  Construct prime in-subgraph $G^0(u|*)$ on G;
  $r_u^0 \leftarrow r^0(u|*)$;
if $\kappa =$ 'O' **then**
  Construct prime out-subgraph $G^0(*|u)$ on G;
  $r_u^0 \leftarrow r^0(*|u)$;
$r_u^{(\eta)} \leftarrow r_u^0$;
if $\eta > 0$ **then**
  for $i = 1 \ldots \eta$ **do**
    $H_i \leftarrow$ hubs in r_u^{i-1};
    foreach $h_i \in H_i$ **do**
      if $\kappa =$ 'I' **then**
        Expand $G^{i-1}(u|h_i)$ with $G^0(h_i|*)$;
        $r_u^i \leftarrow r^0(u|h_i) r^0(h_i|*)$;
      if $\kappa =$ 'O' **then**
        Expand $G^{i-1}(h_i|u)$ with $G^0(*|h_i)$;
        $r_u^i \leftarrow r^0(*|h_i) r^0(h_i|u)$;
    end
    $r_u^{(\eta)} \leftarrow r_u^{(\eta)} + r_u^i$;
  end
return $r_u^{(\eta)}$.

VI. Complexity and error analysis

In this section, we present an analysis of the UISim algorithm, in terms of its complexity and error bound.

A. Complexity analysis

Time analysis. Since one-hop multi-meeting tours correction can be done in constant time, we focus the complexity analysis on 1) initial prime subgraphs construction cost, 2) prime subgraphs expansion cost, and 3) full tours assembling cost.

First, initial prime subgraph construction cost. Depending on the mode of SimRank query, we have different strategies to construct the initial prime subgraphs—prime in-subgraphs of u and v for single-pair query $Q = (u, v)$, prime in-subgraph of u and prime out-subgraphs of corresponding meeting nodes $x \in X_u$ for single-source query $Q = (u, V)$, and prime out-subgraphs of each meeting node $x \in X$ for all-pair query $Q = (V, V)$. If the query nodes or meeting nodes are not hubs, we need to construct their prime subgraphs and compute the reachability of their tours on the fly, which cost $O(Imd)$ using a naïve fixed-point method with I iterations, for a prime subgraph containing m nodes with average degree d.

Second, prime subgraph expansion cost. Prime subgraphs expansion is to extend the initial prime subgraphs at their border hubs (*i.e.*, assemble the precomputed prime subgraphs of each border hub), iteration by iteration, to build the candidate partial tours. Assuming an average degree of d, the sum of degrees of all hubs nodes d_H and the sum of degree of all nodes d_V, there are $\mathcal{T} = O((d(1 - d_H/d_V))^L)$ partial tours of up to length L in each prime subgraph. That is, at each node (starting from the query node), among the d neighbors, $d(d_H/d_V)$ is the number of hub nodes (given that d_H/d_V is the probability of an outgoing edge leading to a hub) where the expansion would stop, and $d(1 - d_H/d_V)$ is the number of non-hubs which will be further expanded to span longer tours. Note that, hub nodes are typically nodes with largest degrees, and thus $1 - d_H/d_V$ would be small. Moreover, when H becomes larger, $1 - d_H/d_V$ and hence $\mathcal{T}$ will decrease—a prime subgraph reduces its size significantly when the number of hubs increases.

Suppose a prime subgraph has $|\bar{H}|$ border hubs. Clearly, $|\bar{H}| \leq |H|$, and in most cases $|\bar{H}| \ll |H|$. In each iteration:

- For *single-pair* mode, we extend $|\bar{H}|$ prime in-subgraphs (each of which contains $\mathcal{T}$ tours) on each side of the query

node.

- For *single-source* mode, we extend $|\bar{H}|$ prime in-subgraphs on the side of the query node, and $|\bar{X}||\bar{H}|$ prime out-subgraphs on the side of meeting nodes $\bar{X}$ for the given query. Clearly $|\bar{X}| \leq |X|$ where X is the set of all meeting nodes.
- For *all-pair* mode, we extend at most $|X||\bar{H}|$ prime out-subgraphs on all meeting nodes X.

Thus, the complexity of η iterations of expansion is bounded by $O(|\bar{H}|^\eta \mathcal{T})$, $O(|\bar{X}||\bar{H}|^\eta \mathcal{T})$ and $O(|X||\bar{H}|^\eta \mathcal{T})$ for the three modes, respectively.

Note that, in UISim we use *iteration* to refer to the extension of subgraphs, which is different from the fixed-point iteration as used in traditional iterative methods. Specifically, the fixed-point iterative method generally stabilizes after 5 iterations [8], which means we only need to handle partial tours of natural length up to 5 (*i.e.*, $L = 5$). In UISim, the hub length of a tour is generally much smaller than its natural length, and thus it is sufficient to cover the necessary tours with only 1–2 expansions (*i.e.*, $\eta \leq 2$), as our experiments in Sect. VII would also confirm.

Lastly, full tour assembling cost. When the candidate partial tours on each side of the meeting nodes are spanned, they will be matched to build the full tours. To generate valid full tours, only the partial tours of the same length will be assembled. Given an expanded subgraph, there are $\mathcal{T}$ partial tours up to length L as discussed earlier. Thus, the cost to match two set of partial tours up to length L in a subgraph is $\mathcal{T}^2$. That is, we have $\mathcal{T}$ tours on each set, and we need to do pair-wise assembling of them. Thus, for single-pair mode where only two subgraphs will be handled, the assembling cost is $O(\mathcal{T}^2)$, for single-source mode, it costs $O(|\bar{X}|\mathcal{T}^2)$ to assemble $|\bar{X}|$ pairs of subgraphs, and for all-pair mode, $O(|X|\mathcal{T}^2)$ is required to assemble $|X|$ pairs of subgraphs.

Space analysis. The space cost depends on the number of prime subgraphs handled in each iteration. Following the time analysis, all the modes require $O(md)$ space for the initial prime subgraphs. In addition, the single-pair mode requires an extra space of $O(md|\bar{H}|^\eta)$ to store the prime subgraphs used in η expansions. Similarly, the single-source and all-pair modes require an extra space of $O(md|\bar{X}||\bar{H}|^\eta)$ and $O(md|X||\bar{H}|^\eta)$, respectively.

Summary. We summarize the time and space complexity analysis in Fig. 5. We make two remarks on the computation of UISim.

First, the three modes of UISim are necessary for efficient mode-specific computation. Comparing across the three modes, we clearly observe that the advantage of mode-specific query processing techniques in terms of both time and space. Specifically, the single-source cost is smaller than that of repeating single-pair queries for $|V|$ times since $|\bar{X}| \ll |V|$, and the all-pair cost is much smaller than repeating the single-pair mode for $|V|^2$ times since $|X| \ll |V|^2$, or repeating the single-source mode for $|V|$ times since $|X| \ll |V||\bar{X}|$.

Mode	Time			Space
	subgraph construction	subgraph extension	full tour assembling	
Single-pair	$O(Imd)$	$O(\lvert\bar{H}\rvert^\eta \mathcal{T})$	$O(\mathcal{T}^2)$	$O(md\lvert\bar{H}\rvert^\eta)$
Single-source		$O(\lvert\bar{X}\rvert\lvert\bar{H}\rvert^\eta \mathcal{T})$	$O(\lvert\bar{X}\rvert\mathcal{T}^2)$	$O(md\lvert\bar{X}\rvert\lvert\bar{H}\rvert^\eta)$
All-pair		$O(\lvert X\rvert\lvert\bar{H}\rvert^\eta \mathcal{T})$	$O(\lvert X\rvert\mathcal{T}^2)$	$O(md\lvert X\rvert\lvert\bar{H}\rvert^\eta)$
		where $\mathcal{T} = (d(1 - d_H/d_V))^L$		

Fig. 5: Time and space analysis of the three modes.

Second, UISim is efficient and scalable. Its time cost is dominated by the prime subgraphs expansion cost (*e.g.*, $O(|\bar{H}|^\eta \mathcal{T})$, where η is typically in $[0, 2]$, and $|\bar{H}| \ll |V|$. More importantly, given more hubs, each prime subgraph handled in computation becomes smaller rapidly. That is, both m and $\mathcal{T}$ significantly decrease with a larger number of hubs. Similarly, the prime subgraph construction and full tour assembling cost also decreases with a larger H. Therefore, UISim is scalable to larger graphs by selecting a large number of hubs. Our experiments in Sec. VII-D validates the scalability of UISim to very large graphs.

B. Error bound analysis

As FastSim incrementally handles partitions of query tours to approximate the SimRank score of any nodes u and v, the accuracy of the approximation improves with more iterations of enhancement. Formally, we establish the following theorem on the expected error after η iterations.

Theorem 1. The expected error in $\hat{s}^{(\eta)}(u, v)$, which represents the SimRank estimation between u and v after η iterations, satisfies the following bound:

$$\mathbb{E}_{u,v \in V}\left[s(u, v) - \hat{s}^{(\eta)}(u, v)\right] \leq \left(\frac{d_H}{d_V}\right)^{\eta+1} C^{\eta+2}, \tag{15}$$

where d_V is the sum of degrees of all nodes, and d_H is the sum of degrees of all hub nodes.

Proof. To compute the expected error, we investigate the length of partial tours covered after η iterations. First, all of the partial tours up to length $\eta + 1$ have been covered. Partial tours of exactly length $\eta + 1$ only accounts for a fraction of $\left(\frac{d_H}{d_V}\right)^\eta$ of all tours starting from the query node. Furthermore, for such a partial tour, there is a probability of $\frac{d_H}{d_V}$ when the partial tour ends at a hub node and thus cannot extend further. Thus, among all the partial tours, a fraction of $\left(\frac{d_H}{d_V}\right)^{\eta+1}$ will not extend to length $\eta + 2$ or longer. In other words, this fraction of the set of partial tours of length $\eta + 2$ or longer are not covered after η iterations. As established previously [18], the total contribution of all $\eta+2$ or longer partial tours is bounded by $C^{\eta+2}$. Thus, in our case, the expected error is bounded by $\left(\frac{d_H}{d_V}\right)^{\eta+1} C^{\eta+2}$. ∎

Since $C < 1$ and $\frac{d_H}{d_V} < 1$, the bound approaches 0 at an exponential rate as η grows. In other words, an earlier iteration contributes exponentially more to the SimRank score. Plugging in some plausible values $C = 0.75$, $\frac{d_H}{d_V} = 0.2$ and

$\eta = 2$, we get the bound as 0.00253, which is fairly tight given that $0 \leq s(u, v) \leq 1$.

VII. Empirical Evaluation

We empirically evaluated UISim on several real-world graphs. The experiments showed that UISim is substantially more efficient than previous state-of-the-art baselines in all three modes, and can also scale to larger graphs.

A. Experimental setup

Datasets. We use eight real-world datasets from different domains and with different properties and sizes summarized in Fig. 6. In particular, six datasets are used for baseline comparison where three smaller graphs of them are also used for parameter study, and two evolving graphs with several snapshots are used to test the scalability of UISim.

Throughout the experiments, we use a typical damping factor of $C = 0.75$.

Environment. We implement all methods in C++, and evaluate them on a Linux system with 3.5GHz CPU and 96GB RAM.

B. Experiments on smaller graphs.

We first evaluate the algorithms on three smaller graphs, 4Area, WikiVote and CondMat (Fig. 6), where the exact SimRank scores can be obtained by the power-iteration method.

Test queries and evaluation. In the single-pair and single-source modes, we randomly sample 100 queries from each graph. Given a query, all the methods compute approximate SimRank scores. Thus, we need to evaluate their accuracy *w.r.t.* the exact scores based on the näive computation. In particular, for single-pair queries, we adopt the metrics of *L1 Similarity* (L1S) and *Relative Goodness* (RG). For a node pair, suppose its exact SimRank score is s and the estimated score is $\hat{s}$. Subsequently, L1S is simply defined as $1 - |s - \hat{s}|$, and RG as $\min\{(s+\delta)/(\hat{s}+\delta), (\hat{s}+\delta)/(s+\delta)\}$ where δ =1E-4 to avoid division by zero. We then report the average of the 100 test pairs for each metric.

For each single-source query, we compute the SimRank scores of other nodes *w.r.t.* the query node, which enable us to obtain a ranking of nodes in decreasing SimRank scores. Given that users are often more interested in first few ranked results, we evaluate the accuracy of top 20 nodes in the ranking. L1S can be similarly employed on the exact and estimated SimRank scores of these 20 result nodes for each query. RG can be extended to measure the "relative goodness" of a ranking, called *Relative Average Goodness* (RAG) as defined previously [31], [32], [2]. As both L1S and RAG evaluate the accuracy of the scores, we additionally use *precision* (Prec) to evaluate the accuracy of rankings, which is the fraction of correct nodes in the top 20. We also average over the 100 test queries for each metric.

In all-pair mode, we initially compute the SimRank scores of all $\frac{1}{2}|V|^2$ node pairs (*i.e.*, there is only one query consisting of all the pairs). Since the vast majority of this enormous number of pairs are uninteresting with very low SimRank scores, we evaluate the accuracy of the top K most similar pairs with largest SimRank scores. K will be set to the number of candidate nodes $|CN|$ used in the all-pair baseline TreeWand [23]. (Note that UISim can compute the SimRank scores of all pairs, not just the $|CN|$ candidates.) We also use L1S, RAG and Prec as our accuracy metrics.

Impacts of different settings. As discussed, we have two main parameters, namely, number of hubs $|H|$ and number of iterations η. We first study their impacts on the performance of UISimand discuss how to set the parameters.

Number of hubs. We first illustrate the effect of varying number of hubs $|H|$ in Fig. 7, where we fix $\eta = 2$. On the one hand, in most scenario having more hubs drastically reduces the average query time of UISim, just as we have expected in Sect. VI. That is, with more hubs H, the number of partial tours in each prime subgraph $\mathcal{T}$ decreases exponentially, and thus both the subgraph extension time and full tour assembling time are decreased. On the other hand, when we have more hubs, we also observe a slight decrease in accuracy as the number of non-hubs which will be further expanded to span longer tours decreases. That is, more expansions are stopped by the border hubs, potentially hurting accuracy. Nevertheless, as we reasonably increase $|H|$ in Fig. 7, most drops in accuracy are very minor while query processing becomes much faster, which is consistent with our theoretical analysis in Sect. VI. That is, when $|H|$ becomes larger, the decrease in running time is exponential while the increase in expected error is linear. Thus, it is still beneficial to use a relatively large $|H|$.

In practice, we should also consider the structure of graphs (*e.g.*, d_H and d_V in Eq. 15) to set the value $|H|$. More hubs should be selected on larger and denser graphs. Thus, to determine the number of hubs, a simple rule is $|H| = \beta \log(d)|V|$, where d is the average node degree for some choice of $\beta > 0$. Empirically, the desirable range of β is between 0.1 and 0.5 for a reasonable trade-off between accuracy and time.

Number of iterations. Next, we study the ability of *incremental* query processing by UISim. We vary the number of expansion iterations η in Fig. 8, where we fix $|H|$. Our results show that more iterations result in better accuracy (if not already good at $\eta = 0$), but require longer time to process. Thus, the accuracy of our SimRank estimation indeed improves in an incremental manner. In particular, accuracy improvement is generally more significant in earlier iterations (from $\eta = 0$ to 1 as compared to from $\eta = 1$ to 2). In most cases, high accuracy can be obtained with very few iterations at $\eta = 1$.

Comparison to baselines. We compare UISim with the state-of-the-art competitors in each query mode: BLPMC, the index-free single-pair solution [24], ProbeSim, the single-source solution [16] and TreeWand, the all-pair solution [23].

As all algorithms compute approximate SimRank scores, there is a trade-off between accuracy and query time. In order to fairly compare different methods, we should fix their accuracy at a similar level, and then compare the running

Dataset	Description	Directed	Nodes	Edges	Purpose
4Area	DBLP bibliographic network in four areas, similar to ref. [22], [4]	no	12 413	91 192	Parameter study, and comparison to baselines (with ground truth)
WikiVote	Wikipedia administrator election network [12] (dangling nodes removed)	yes	1 300	39 456	
CondMat	Collaboration network of Arxiv Condensed Matter [12]	no	23 133	93 497	
enwiki2013	A snapshot of the English part of Wikipedia[12] [1]	yes	4 206 785	101 355 853	Comparison to baselines (without ground truth)
it2014	A fairly large crawl of the .it domain [1]	yes	41 291 594	1 150 725 436	
Friendster	On-line gaming network [12]	yes	65 608 366	1 806 067 135	
Gnutella	Gnutella peer to peer network with several snapshots [12]	yes	62 586	147 892	Scalability study
Dblp	Full DBLP bibliographic network with several snapshots, similar to ref. [4]	no	1 183 496	18 828 154	

Fig. 6: Summary of datasets.

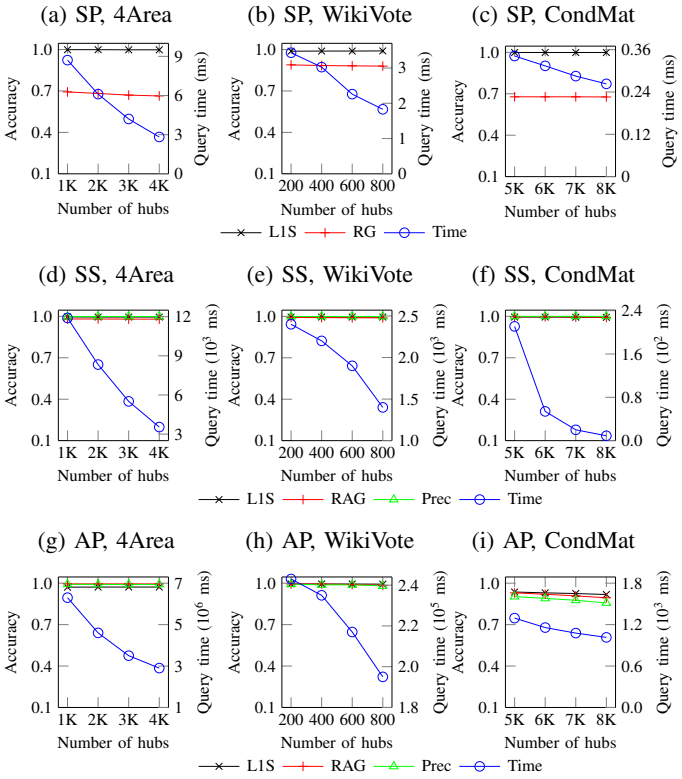

Fig. 7: Impact of number of hubs on accuracy metrics (left y-axis) and query time (right y-axis) in three modes: single pair (SP), single source (SS) and all pair (AP).

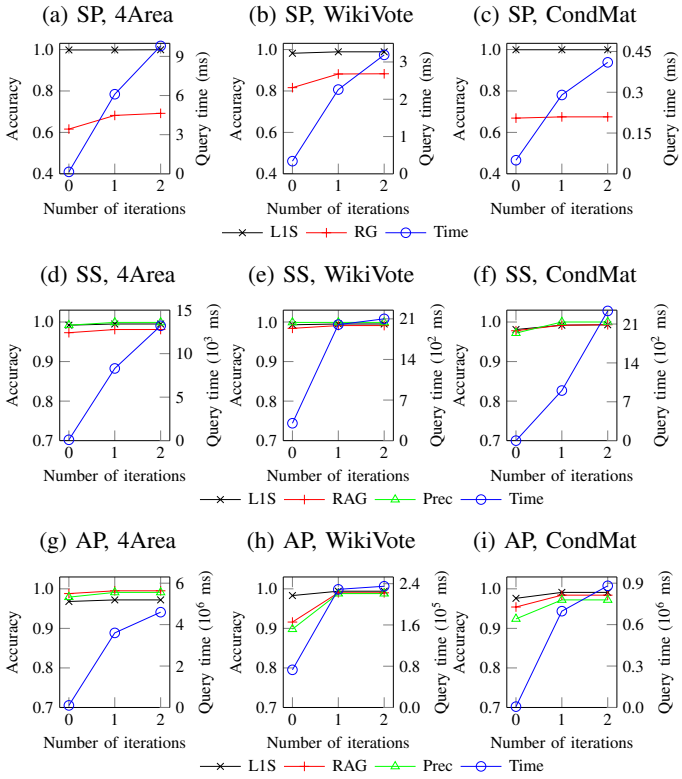

Fig. 8: Impact of number of iterations on accuracy metrics (left y-axis) and query time (right y-axis) in three modes: single pair (SP), single source (SS) and all pair (AP).

time under these settings. To obtain comparable accuracy, the parameter settings in different methods cannot be directly derived from their theoretical error bounds, which have different formulations (*e.g.*, some are deterministic, some are probabilistic, and some are in the expectation sense), and have varying degrees of tightness. Therefore, to systematically compare different methods in practice, we vary the parameters of each method in a reasonably large range, so as to evaluate a large number of different configurations. Subsequently, we compare different methods under these configurations that give similar accuracy.

Parameter setting. For all algorithms, we set the damping factor $C = 0.75$; for both BLPMC and ProbeSim, we set the failure probability of the Monte Carlo simulation $\delta \in [10^{-9}, 10^{-1}]$ as suggested in [24]. Specifically, ϵ, the error bound in BLPMC is varied from 0.005 to 0.015 at the step of 0.001; ε_a, the maximum absolute error in ProbeSim is set to $\varepsilon = \{0.1, 0.01, 0.001\}$; δ_k, the accuracy loss is set to $\delta_k = \{10^{-2}, 10^{-3}, 10^{-4}\}$, and $|CN|$, the number of candidate nodes in TreeWand ranges from 1000 to 5000 at the step of 1000. Other parameters are specified according to the original papers. For UISim we vary $|H|$ in the range discussed earlier, and for each value of $|H|$ we try $\eta = \{0, 1, 2\}$.

We run all the settings and evaluate their accuracy using the metrics explained earlier. For single-pair and single-source mode, we plot the results of settings with running time falling into a same range in Fig. 9, where x-axis the is running time and y-axis is the accuracy. Here we only present L1S for the single-pair mode and precision for the single-source mode as the accuracy metric, since we observe similar trends in other metrics as well. Typically, we focus on relatively small running time (*e.g.*, from 0 to 20 ms in the single-source mode) as

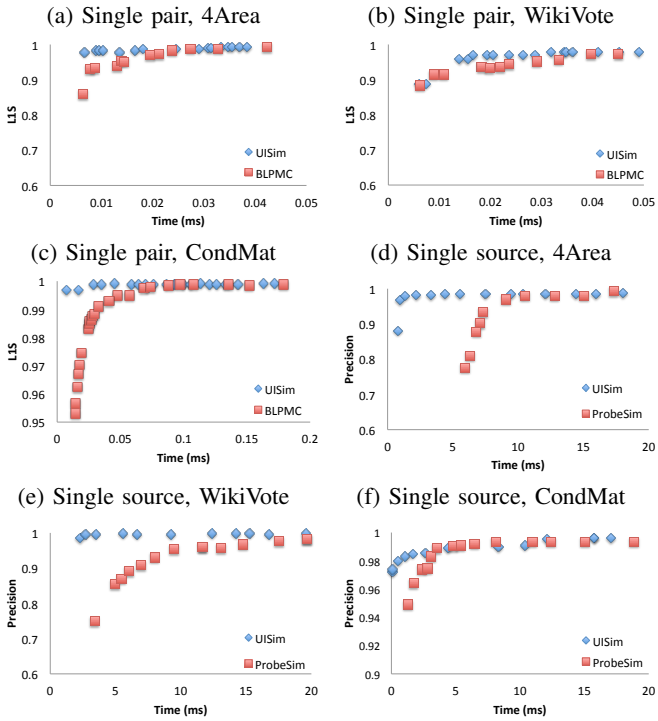

Fig. 9: Comparison of accuracy against time with baselines in single pair and single source modes.

Dataset	L1S		RAG		**Time (ms)**	
	UISim	BLPMC	UISim	BLPMC	UISim	BLPMC
4Area	.978	.954	.563	.570	0.007	0.014
	.984	.969	.571	.531	0.009	0.019
	.990	.983	.648	.648	0.01	0.024
WikiVote	.960	.939	.781	.815	0.014	0.033
	.961	.950	.781	.790	0.016	0.029
	.977	.972	.871	.882	0.031	0.045
CondMat	.999	.944	.666	.660	0.007	0.013
	.999	.974	.666	.663	0.009	0.019
	.999	.995	.669	.671	0.017	0.048

Fig. 10: Detailed comparison in single-pair mode.

many applications will require a fast online computation of SimRank. For each method, if we observe different accuracy with the same running time, we report the highest accuracy.

From Fig. 9, we can clearly observe the advantage of UISim. On the one hand, UISim always need less time to achieve the same accuracy as its baseline. For a more concrete comparison, we further list several configurations in Fig. 10 and 11 with detailed accuracy and running time. As observed, to achieve similar accuracy, UISim runs faster (up to 1–2 orders of magnitude) than the respective baseline in each mode. On the other hand, UISim can achieve a good accuracy very fast while the baseline does not perform well within limited time. For example, as shown in Fig. 9(d)(e), UISim can achieve accuracy above 0.95 within 5ms, while given the same time, the best accuracy of ProbeSim is around 0.85. Such observation also validates the benefit of the "important-first" property of UISim discussed in Sect. I.

For all-pair SimRank, since running one configuration can take several or even hundreds of hours for the baseline

Dataset	L1S		RAG		Prec		**Time (ms)**	
	UISim	ProbeSim	UISim	ProbeSim	UISim	ProbeSim	UISim	ProbeSim
4Area	.953	.953	.937	.915	.937	.933	2.8	7.3
	.952	.952	.959	.950	.984	.981	3.6	10.5
	.950	.950	.981	.993	.999	1	2047.0	36394.0
WikiVote	.989	.957	.975	.907	.995	.854	2.4	5.0
	.950	.950	.981	.985	.998	.998	87.7	166.0
	.950	.950	.991	.990	.999	.999	2327.0	8476.0
CondMat	.981	.978	.978	.969	.972	.972	0.3	1.5
	.987	.978	.985	.979	.988	.988	1.7	1.8
	.994	.978	.995	.994	1	1	196	9729.0

Fig. 11: Detailed comparison in single-source mode.

Dataset	L1S		RAG		Prec		**Time (hr)**	
	UISim	TreeWand	UISim	TreeWand	UISim	TreeWand	UISim	TreeWand
4Area	.971	.970	.995	.995	.992	.991	1.590	112.400
	.937	.926	.967	.963	.942	.936	0.600	48.640
	.658	.644	.578	.580	.578	.580	0.010	35.030
WikiVote	.996	.998	1	1	1	1	0.060	0.200
	.994	.995	.984	.973	.980	.965	0.060	0.370
	.995	.996	.994	.990	.993	.988	0.070	0.680
CondMat	.959	.895	.866	.828	.813	.779	0.003	0.007
	.924	.929	.913	.909	.897	.893	0.003	0.015
	.935	.925	.940	.940	.924	.919	0.006	1.911

Fig. 12: Detailed comparison in all-pair mode.

TreeWand, and the range of time we observed in UISim and TreeWand are vastly different, we only report the detailed results of nine configurations in Fig. 12. The results show that UISim is faster than the TreeWand by 1 to 2 orders of magnitude, and at the same time achieves similar or better accuracy levels.

C. Experiments on larger graphs

We next evaluate UISim on three large graphs Enwiki2013, IT2004 and Friendster, with up to 65 million nodes and 1.8 billion edges (Fig. 6). We focus on the single-source node comparison with the baseline ProbeSim, since for single-pair queries, the baseline method runs out of memory on all the three graphs. All-pair mode is also not evaluated here, as most applications would not need all-pair results on graphs of this scale.

On large graphs, it is impractical to calculate the exact SimRank similarities to evaluate the accuracy of different algorithms. However, we can still compare the similarity of the results produced by UISim and ProbeSim. Thus, as long as their results are adequately similar, it is still reasonable to compare their running time, even though we do not know their actual accuracy. In particular, we take the top-K nodes from the output rankings of each method. For these K nodes, we use RAG to measure the average similarity of the scores estimated by the two methods, and the Jaccard similarity to evaluate the overlap between the two methods. We choose $K = 20$ in our experiment.

In particular, to obtain the similar results for different methods, we first set $e = 0.001$ in ProbeSim and $\eta = 1$ in UISim which have demonstrated good accuracy on smaller datasets. We then try different values of $|H|$ in UISim to ensure that UISim produces top-K results similar to ProbeSim's, as evaluated by RAG and Jaccard.

Dataset	Parameters		Similarity of results		**Time (s)**	
	UISim ($\eta = 1$)	ProbeSim	RAG	Jaccard	UISim	ProbeSim
Enwiki2013	$\lvert H\rvert = 3000K$	$e = 0.001$	0.948	0.865	14	117128
IT2004	$\lvert H\rvert = 9000K$	$e = 0.001$	0.733	0.650	306	78708
Friendster	$\lvert H\rvert = 5000K$	$e = 0.001$	0.974	0.870	154285	1022304

Fig. 13: Average single-source query time on large graphs.

	# Nodes	# Edges	# Hubs	Avg. query time (ms)	
	$\lvert V\rvert$	$\lvert E\rvert$	$\lvert H\rvert$	Single pair	Single source
Gnutella1	6 301	20 777	600	0.6	23
Gnutella2	8 717	31 525	1 000	0.6	48
Gnutella3	10 876	39 994	1 500	0.6	50
Gnutella4	22 687	54 705	2 000	1.2	25
Gnutella5	36 682	88 328	3 000	1.5	42
Gnutella6	62 586	147 892	8 000	4.4	64
Dblp1	105 903	1 165 716	2 000	10	5 410
Dblp2	232 158	2 842 032	5 000	21	7 833
Dblp3	439 577	5 865 472	8 000	39	15 928
Dblp4	818 219	11 971 444	10 000	71	53 069
Dblp5	1 183 496	18 828 154	15 000	112	77 614

Fig. 14: Scalability of UISim on growing graphs.

Fig. 13 shows the comparison results on the three large graphs. On each graph, 10 random queries are chosen and their average results are reported. It can be observed that, UISim is at least 10 times faster than ProbeSim, while they produce similar results with reasonably high RAG and Jaccard scores.

D. Scalability on growing graphs

Finally, we test the scalability of UISim on two growing graphs Gnutella and Dblp (Fig. 6). In particular, Gnutella includes snapshots on different dates, whereas Dblp includes snapshots of different years. Each subsequent snapshot has a larger number of nodes $|V|$ and edges $|E|$, as shown in Fig. 14.

The key to scaling UISimis to increase the number of hubs $|H|$. As discussed in Sect. VII-B, using more hubs can reduce query time. Thus, we use a larger $|H|$ on a larger snapshot. As shown in Fig. 14, by using more hubs, we are able to achieve an approximate *linear* scale-up of query time in single-pair and single-source modes. That is, when we double the size of the graph, the average query time roughly doubles too.

VIII. Conclusion

In this paper, we presented a unified framework to efficiently process all three modes of SimRank. As our key principle, we conceptually scheduled the tours for a prioritized computation, which exhibits two desirable properties: "important-first" and "incrementally-enhanced." To realize this principle, we developed a benefit-based tour assembling model and mode-specific tour spanning and matching techniques to effectively process each mode of queries. Empirically, UISim is not only superior to the strongest baselines designed specifically for each mode, but also scalable to larger graphs.

Appendix

Recall that in Sect. III, we relax the *first-meeting* constraint in partial-tour assembling (Eq. 4), and thus the *multi-meeting tours* could be included in computation. We now discuss how to correct the reachability of the unexpected multi-meeting tours.

Consider a set of full tours $T_{(u,v)} = P_u \bowtie P_v$ currently handled in estimating $\hat{s}(u,v)$. For any meeting node $x \in X$, if it has an in-neighbor x_1, there will be a multi-meeting tour $u \nleftrightsquigarrow x \leftarrow x_1 \rightarrow x \rightsquigarrow v$ in $T_{(u,v)}$, where the unexpected meeting node x_1 is *1 hop* away from the first-meeting node x. We refer to such tours as *1-hop multi-meeting tours*. Furthermore, there could be *2-hop multi-meeting tours* $u \nleftrightsquigarrow x \leftarrow y \leftarrow x_2 \rightarrow z \rightarrow x \rightsquigarrow v$ with the unexpected meeting node x_2 *2 hops* away from x, and so on. Similarly, x_2 can have its own in-neighbors, which results in a set of *three-hop multi-meeting tours.*

To correct $\hat{s}(u,v)$, the estimated SimRank on $T_{(u,v)}$, we should subtract the reachability of all the multi-meeting tours, from 1-hop to M-hop (M is the maximal tour length). Let $\Delta_i(u,v)$ denote the overall reachability of i-hop multi-meeting tours between u and v, we can obtain the correct SimRank score $s(u,v)$ as:

$$s(u,v) = \hat{s}(u,v) - \sum_{i=1}^{M} \Delta_i(u,v) \tag{16}$$

Let us start with 1-hop multi-meeting tours. We notice that all such tours share the same type of tour segment $x \leftarrow x' \rightarrow x$, that is, $R(x \leftarrow x' \rightarrow x)$ is a common factor in computing the reachability of any 1-hop multi-meeting tour. Thus, to calculate $\Delta_1(u,v)$, we can segment each 1-hop multi-meeting tour into two parts, $x \leftarrow x' \rightarrow x$ and $u \nleftrightsquigarrow x \rightsquigarrow v$, and then compute the overall reachability of the tour segments in each part respectively to assemble the final reachability, as formalized below.

Proposition 1. For any nodes u and v, $\Delta_1(u,v)$, the overall reachability of 1-hop multi-meeting tours in $T_{(u,v)}$ is:

$$\Delta_1(u,v) = \sum_{x \in X} \frac{C}{|In(x)|} \hat{s}_x(u,v). \tag{17}$$

where $\hat{s}_x(u,v)$ is the overall reachability of all the tours which meet at x in $T_{(u,v)}$. ∎

Proof. According to Eq. 6, the reachability of a specific 1-hop multi-meeting tour $t_x : u \nleftrightsquigarrow x \leftarrow x_1 \rightarrow x \rightsquigarrow v$ is $R(t_x) = \frac{C}{|In(x)|^2} R(u \nleftrightsquigarrow x \rightsquigarrow v)$. Let Ψ_x denote the set of 1-hop multi-meeting tours *w.r.t.* x in $T_{(u,v)}$, we have

$$R(\Psi_x) = \sum_{x' \in In(x)} R(t_{x'}) = |In(x)| \times \frac{C}{|In(x)|^2} \hat{s}_x(u,v) = \frac{C}{|In(x)|} \hat{s}_x(u,v). \tag{18}$$

By further aggregating $R(\Psi_x)$ of every x, the overall reachability of all 1-hop multi-meeting tours in $T_{(u,v)}$ is: $\sum_{x \in X} \frac{C}{|In(x)|} \hat{s}_x(u,v)$. ∎

Based on Proposition 1, the SimRank similarity after correcting $\Delta_1(u,v)$ is calculated as

$$\hat{s}(u,v) - \Delta_1(u,v) = \sum_{x \in X} \hat{s}_x(u,v) - \Delta_1(u,v) = \sum_{x \in X} \left(1 - \frac{C}{|In(x)|}\right) \hat{s}_x(u,v). \tag{19}$$

Thus, to correct the reachability of 1-hop multi-meeting tours, we only need to multiple our estimate by a coefficient

$1 - \frac{C}{|In(x)|}$ when matching the full tours at each meeting node x.

For other multi-meeting tours, their reachability can be computed similarly by utilizing the in-degree of the intermediate nodes between meeting nodes. For example, the overall reachability of 2-hop multi-meeting tours (*i.e.*, tours in the form $u \leftsquigarrow x \leftarrow y \leftarrow x_2 \rightarrow z \rightarrow x \rightsquigarrow v$) is calculated as:

$$\Delta_2(u, v) = \sum_{x \in X} \frac{C^2}{|In(x)|^2} \sum_{y,z \in In(x)} \frac{|In(y) \cap In(z)|}{|In(y)||In(z)|} \hat{s}_x(u, v) \qquad (20)$$

Although it becomes complicated to examine more nodes, it is also clear that those multi-hop tours are insignificant in SimRank estimation, only resulting in a very small difference in the overall score. Thus, as an approximation, we can ignore the other multi-meeting tours, and focus on correcting 1-hop multi-meeting tours only as applied to UISim and TreeWand in our experiments.

References

[1] P. Boldi, A. Marino, M. Santini, and S. Vigna. BUbiNG: Massive crawling for the masses. In *WWW Companion*, pages 227–228, 2014.

[2] S. Chakrabarti. Dynamic personalized pagerank in entity-relation graphs. In *WWW*, pages 571–580, 2007.

[3] L. Du, C. Li, H. Chen, L. Tan, and Y. Zhang. Probabilistic simrank computation over uncertain graphs. *Inf. Sci.*, 295:521–535, 2015.

[4] Y. Fang, K. C. Chang, and H. W. Lauw. Roundtriprank: Graph-based proximity with importance and specificity? In *ICDE*, pages 613–624, 2013.

[5] D. Fogaras and B. Rácz. Scaling link-based similarity search. In *WWW*, pages 641–650, 2005.

[6] Y. Fujiwara, M. Nakatsuji, H. Shiokawa, and M. Onizuka. Efficient search algorithm for SimRank. In *ICDE*, pages 589–600, 2013.

[7] G. He, H. Feng, C. Li, and H. Chen. Parallel simrank computation on large graphs with iterative aggregation. In *KDD*, pages 543–552, 2010.

[8] G. Jeh and J. Widom. SimRank: a measure of structural-context similarity. In *KDD*, pages 538–543, 2002.

[9] M. Jiang, A. W.-C. Fu, and R. C.-W. Wong. Reads: A random walk approach for efficient and accurate dynamic simrank. *PVLDB*, 10(9):937–948, May 2017.

[10] M. Kusumoto, T. Maehara, and K.-i. Kawarabayashi. Scalable similarity search for SimRank. In *SIGMOD*, pages 325–336, 2014.

[11] P. Lee, L. V. S. Lakshmanan, and J. X. Yu. On top-k structural similarity search. In *ICDE*, pages 774–785, 2012.

[12] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[13] C. Li, J. Han, G. He, X. Jin, Y. Sun, Y. Yu, and T. Wu. Fast computation of SimRank for static and dynamic information networks. In *EDBT*, pages 465–476, 2010.

[14] P. Li, H. Liu, J. X. Yu, J. He, and X. Du. Fast single-pair SimRank computation. In *SDM*, pages 571–582, 2010.

[15] Z. Li, Y. Fang, Q. Liu, J. Cheng, R. Cheng, and J. C. S. Lui. Walking in the cloud: Parallel simrank at scale. *PVLDB*, 9(1):24–35, 2015.

[16] Y. Liu, B. Zheng, X. He, Z. Wei, X. Xiao, K. Zheng, and J. Lu. Probesim: Scalable single-source and top-k simrank computations on dynamic graphs. *PVLDB*, 11(1):14–26, 2017.

[17] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation. *PVLDB*, 1(1):422–433, 2008.

[18] D. Lizorkin, P. Velikhov, M. Grinev, and D. Turdakov. Accuracy estimate and optimization techniques for simrank computation. *PVLDB*, 19(1):45–66, 2010.

[19] T. Maehara, M. Kusumoto, and K. Kawarabayashi. Efficient simrank computation via linearization. In *KDD*, pages 1426–1435, 2014.

[20] T. Maehara, M. Kusumoto, and K.-i. Kawarabayashi. Scalable simrank join algorithm. In *ICDE*, pages 603–614, 2015.

[21] Y. Shao, B. Cui, L. Chen, M. Liu, and X. Xie. An efficient similarity search framework for SimRank over large dynamic graphs. *PVLDB*, 8(8), 2015.

[22] Y. Sun, J. Han, J. Gao, and Y. Yu. iTopicModel: Information network-integrated topic modeling. In *ICDM*, pages 493–502, 2009.

[23] W. Tao and G. Li. Efficient top-k SimRank-based similarity join. In *SIGMOD*, pages 1603–1604, 2014.

[24] Y. Wang, L. Chen, Y. Che, and Q. Luo. Accelerating pairwise simrank estimation over static and dynamic graphs. *PVLDB*, 28(1):99–122, 2019.

[25] Y. Wang, X. Lian, and L. Chen. Efficient simrank tracking in dynamic graphs. In *ICDE*, pages 545–556, 2018.

[26] W. Yu, X. Lin, and J. Le. A space and time efficient algorithm for SimRank computation. In *APWeb*, pages 164–170, 2010.

[27] W. Yu, X. Lin, and J. Le. Taming computational complexity: Efficient and parallel simrank optimizations on undirected graphs. In *WAIM*, pages 280–296, 2010.

[28] W. Yu, X. Lin, and W. Zhang. Towards efficient SimRank computation on large networks. In *ICDE*, pages 601–612, 2013.

[29] W. Yu and J. A. McCann. Sig-sr: SimRank search over singular graphs. In *SIGIR*, pages 859–862, 2014.

[30] W. Yu and J. A. McCann. Efficient partial-pairs SimRank search on large networks. *PVLDB*, 8(5), 2015.

[31] F. Zhu, Y. Fang, K. C.-C. Chang, and J. Ying. Incremental and accuracy-aware personalized pagerank through scheduled approximation. *PVLDB*, 6(6):481–492, 2013.

[32] F. Zhu, Y. Fang, K. C.-C. Chang, and J. Ying. Scheduled approximation for personalized pagerank with utility-based hub selection. *VLDBJ*, pages 1–25, 2015.

[33] R. Zhu, Z. Zou, and J. Li. Simrank computation on uncertain graphs. In *ICDE*, pages 565–576, 2016.