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

Fanwei Zhu, Yuan Fang, Kai Zhang, Kevin Chen-Chuan Chang, Hongtai Cao, Zhen Jiang, and Minghui Wu

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, and scales well on larger graphs.

Index Terms—SimRank approximation, unification, index-free, scheduled principle, scalability.

1 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. [6]. 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.

Scenario 1 (Single-pair SimRank): Collaboration prediction. Given two researchers r_1 and r_2, could r_1 be collaborated with r_2 in the future? In this case, we can compute a SimRank score $s(r_1, r_2)$ and compare it with some heuristic threshold to make a prediction.

Scenario 2 (Single-source SimRank): Bibliographic search. Given a paper p, what are the most relevant papers to p in the entire corpus? In this case, the input query is a paper p, and the output is a ranking over all the paper nodes according to the SimRank similarity between p and each paper in the network.

Scenario 3 (All-pair SimRank): Research community discovery. What are the similar papers, researchers, and conferences that form a community of certain research interest? In this case, the similarity between each pair of nodes should be computed and further leveraged by clustering methods such as K-means to detect the research communities in the graph.

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 (Scenario 1); *single-source* SimRank computes the similarity score between a query node u and every node $v \in V$ (Scenario 2); *all-pair* SimRank computes the similarity for every pair of nodes in G (Scenario 3).

TABLE 1: SimRank Problems on a Graph $G = (V, E)$.

SimRank Problems		Query Q=(A,B)	Output
General definition	Partial-pair	$A \subseteq V$ $B \subseteq V$	$s(u, v)$: a A-by-B similarity matrix, with each entry $[S]_{u,v} = s(u, v)$
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 $[S]_{u,v} = s(u, v)$
	All-pair	A=V B=V	$[S]$: a $\|V\|$-by-$\|V\|$ similarity matrix, with each entry $[S]_{u,v} = s(u, v)$

As a general form of SimRank problems, *partial-pairs* SimRank (or SimRank-based Similarity Join) [29], [21], [17], [30], [13], is defined over two subsets of nodes where only similarities between node-pairs from those subsets are computed. Most of existing partial-pair SimRank focus on the subset of node-pairs that have higher similarity than the others in a graph, either returning the top-K similar node-pairs [21], [13] or the node-pairs with similarities greater than a given threshold [17], [30]. More discussions about SimRank Join can be found in Sect. 2.

We summarize these SimRank modes in Table 1, where a SimRank query is formalized 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.

While SimRank is confirmed to be an effective similarity measure in practical applications [7], the computation of SimRank is not trivial. 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

- *F. Zhu, M. Wu, K. Zhang, and Z. Jiang are with Zhejiang University City College, China. E-mail:{zhufw,mhwu}@zucc.edu.cn, drogozhang@gmail.com, jzjzjzzju@zju.edu.cn*
- *Y. Fang (corresponding author) is with Singapore Management University, Singapore. E-mail:yfang@smu.edu.sg*
- *H. Cao and K. Chang are with University of Illinois at Urbana-Champaign, USA. E-mail:{hongtai2, kcchang}@illinois.edu*

based on their in-neighbors $In(u)$ and $In(v)$, as follows [6].

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

Due to the iterative nature, the computation is expensive even on a moderately large graph. Thus, many works have devoted to speedup SimRank computation with approximation. In this paper, we also focus on the efficient approximation of SimRank. 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. 2.

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 [14] and PRSim [26] the state-of-the-art methods based on Monte Carlo simulation, sample 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– 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. For example, in the *Bibliographic search* scenario, a fast estimate of SimRank similarities is expected to quickly return a ranked list of relevant papers, while in the *Collaboration prediction* scenario, more accurate SimRank scores would be preferred for an effective prediction. 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 incrementally-enhanced framework, UISim, to efficiently process different modes of SimRank queries based on the *random surfer-pair model* [6] where the SimRank similarity $s(u,v)$ is 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 is 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 scheduled approximation is deterministic, intentionally prioritized and incrementally enhanced, 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 (*i.e.*, hub 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.

Realization challenges. Note that, the scheduled approximation of UISim shares similar insight with a previous work FastPPV [31] which efficiently handles PPV queries by arranging the important tours first for a fast estimate, as there is a fundamental connection between SimRank and PPV computation– both can be conceptualized as the incremental aggregation of random walk tours with varying importance. However, realizing such principle in SimRank setting posts unique challenges 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 tours $u \leftsquigarrow x \rightsquigarrow v$ that meet at any common node x, while the principle in FastPPV is originally designed for regular tours $u \rightsquigarrow v$ from one node to the other. Simply adapting FastPPV to incrementally expand the regular tours on each side of meeting nodes waste a lot of computations as most of the spanned tours would not have the same length or ending node and thus can not be assembled as valid SimRank tours. One the other hand, the scheduled approximation principle of FastPPV can only ensure the one-side regular tours are partitioned and incrementally processed by their importance, while assembling two sets of important regular tours may not necessarily result in an important partition of two-side SimRank tours as the number of valid assembling from those tours are not guaranteed. Thus, to incrementally explore the computation space of SimRank queries for an important-first approximation, we need to develop new techniques to prioritize the generation of the two-side query tours $T_Q = T_Q^0 \cup \cdots \cup T_Q^\eta$ from regular one-side tours on G.
- *Second, diverse query modes.* SimRank has a variety of concrete modes, each of which has its own requirement to identify the computation space, while FastPPV only solves one kind of single-source query. Extending FastPPV to other modes is problematic. For example, for single-pair queries, spanning of

tours from a source to all reachable nodes would be wasteful as many of the spanned tours would not reach the specific target node; while for all-pair queries, redundant computation over shared tours spanned from different source nodes should be avoided. 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, 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 $P_A \bowtie P_B$ in Sect. 3. 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. 4. 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. 5. We analyze the complexity and error bound of UISim in Sect. 6.

Empirical evaluation. We conduct extensive experiments on various real-world datasets in Sect. 7. We empirically study the effect of parameters in UISim, and compare it with the state-of-the-art baselines in different modes, and find out UISim significantly outperforms its respective baselines in each mode– 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.

2 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.

Iterative methods. Some early approaches directly optimize the basic iterative algorithm, by reducing unnecessary computation and reusing shared computation both within and across iterations. Lizorkin et al. [15] propose to memorize the reusable partial sums across iterations to prevent repeated computation for all-pair SimRank. Yu et al. [27] further reduce the redundancy in computing partial sums with sub-summation sharing in all-pair mode. Li et al. [12] employ position probability to reduce the computation not relevant to a query in the single-pair mode. However, even the state-of-the-art iterative methods [27], [12] 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. [11] derive a linear system and performs singular value decomposition (SVD) on the similarity matrix to get SimRank approximation. Fujiwara et al. [5] propose SimMat that computes SimRank based on the Sylvester equation and low-rank approximation of the similarity matrix. Yu et al. [28] relax 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] propose 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. [4] apply MC simulation to sample random walk paths between two nodes, which addresses single-pair SimRank. Kusumoto et al. [8] later extend it to address the single-source mode through extensive pruning. 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. 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. Wei et al. propose an index-free single-source algorithm ProbeSim [14] that performs MC simulation to sample the $\sqrt{c}-walk$ of query node, and then from each visited node probes the $\sqrt{c} - walk$ on the other side to compute the probability of walk pairs. The authors also propose an index-based algorithm PRSim [26] that decomposes a SimRank query $s(u, v)$ into two $l - hop$ RPPR (reversed Personalized PageRank values) and a last meeting probabilities, combining MC sampling and local push techniques to solve the decomposed computations with precomputed RPPRs. Wang et al. [23] further combine the ideas of PRSim and linearization to derive a probabilistic exact single-source SimRank algorithm ExactSim with additive error of at most $\epsilon_{min} = 10^{-7}$, which can be used to compute the ground truth on billion-edge graphs. Instead of sampling the random walk tours of a query, a recent work SimPush [19] proposes to focus on the query tours around a small set of attention nodes in the close vicinity of the query node to answer single-source SimRank queries. Although ignoring tours around non-attention nodes reduces the computation overhead, it can also hurt the accuracy of approximation. Moreover, in order to select the attention nodes, SimPush needs to compute the hitting probabilities for all nodes in a source graph– a lot of computations are wasted on non-attention 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. Extending the algorithms designed for one mode to other modes is not feasible. For example, if we adapt the single-source solution ProbeSim or PRSim to answer a single-pair query $s(u, w)$, most of the probes or backward walks would be wasted as they may not hit the specific node w. Although some previous work [29], [16] also address different modes of SimRank, their techniques are essentially designed for certain modes. In particular, Yu et al. [29] conceptually integrate different modes of SimRank problems by a general definition partial pair SimRank (*i.e.*, SimRank similarity between any two sets of nodes). However, they develop an optimized technique for single-source SimRank only, and proposes to decompose the partial-pair problem into multiple single-source problems. Maehara et al. [16] propose a linearized technique to efficiently tackle single-pair and single-source, while the all-pair problem is solved by trivially repeating the single-source solution. To answer threshold-based SimRank Join queries, the authors further propose a filter-and-verification framework [17] to prune the node-pairs based on their SimRank bounds in the filter phase; and assess the similarity of the candidate pairs in the verification phase.

It is worth noting, although it is not tailored for SimRank Join, UISim naturally supports top-K SimRank Join due to its important-first nature– the most similar node-pairs would always be computed earlier as the tours between them have higher importance to be scheduled earlier in our incremental processing framework. On the contrary, existing SimRank Join algorithms mainly rely on some pruning techniques to find a candidate set of promising nodes for further verification given a specific similarity threshold [17], [30] or the number of expected results [13], [21], which is less flexible than the prioritized and incrementally-enhanced approximation of UISim.

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, TopSim [9] and SimPush [19] are also based on path enumeration rather than random simulation. However, they only consider random walk tours in a fixed-length neighborhood of query node, or around some attention nodes in the neighborhood. On the contrary, UISim allows a wide range of tradeoff of time and accuracy by gradually cover the tours in the entire computation space. Another line of local push based algorithms [24], [25] have the same issue that a certain amount of local push operations are required to explore the useful query tours. Therefore, when the time budget is limited, the performance of local push based algorithms is significantly inferior than that of UISim.

Third, in terms of applications, UISim is capable of handling both static and dynamic graphs. Different from the index-based algorithms which needs expensive cost to update their index on dynamic graph [18], [24], 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 proposed to support dynamic updates [9], [14], but UISim outperforms them in 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.

3 Unified Computation space: aggregating tours

As motivated in Sect. 1, 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 the general SimRank queries. To illustrate, we introduce a toy graph G in Fig. 1, and in Fig. 2 we list the query tours of an example partial-pair query $Q = (\{a\}, \{b, g\})$. We observe that the computation space of Q is composed of a set of two-side tours ended with node a and nodes b, g on each side, which can be partialized into two sets of regular tours at the centered nodes x_1, x_2, x_3.

Conceptual view of computation space. Conceptually, we can model the computation space of any SimRank query $Q = (A, B)$ as

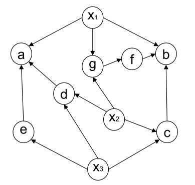


Fig. 1: A toy graph G.

Two-side Query Tours A={a}, B={b, g}	Partial Tours ending at A	Partial Tours ending at B
$a \leftarrow x_1 \rightarrow b$	$a \leftarrow \boxed{x_1}$	$\boxed{x_1} \rightarrow b$
$a \leftarrow x_1 \rightarrow g$		$\rightarrow g$
$a \leftarrow d \leftarrow x_2 \rightarrow c \rightarrow b$	$a \leftarrow d \leftarrow \boxed{x_2}$	$\boxed{x_2} \rightarrow c \rightarrow b$
$a \leftarrow d \leftarrow x_3 \rightarrow c \rightarrow b$	$a \leftarrow d \leftarrow \boxed{x_3}$	$\boxed{x_3} \rightarrow c \rightarrow b$
$a \leftarrow e \leftarrow x_3 \rightarrow c \rightarrow b$	$a \leftarrow e \leftarrow$	

Fig. 2: Query tours of $Q = (\{a\}, \{b, g\})$.

a set of *query tours* T_Q *w.r.t.* the query nodes A and B, formalized as:

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

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

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, $u \leftrightsquigarrow x \rightsquigarrow v$, which end with u and v on each side [12]:

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

It is worth noting that if the graph contains cycles or self-loops, the computation space of certain queries would consist of infinite number of tours. However, a large number of longer tours containing circles would have trivial contribution to the overall score. In other words, although in this cases the exact SimRank scores should be computed over a finite set of query tours, we can still obtain a good approximation over a smaller set of important tours.

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:

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

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, similar as previous works [18], [21], [22], [26]. Including the *multi-meeting* tours (*i.e.*, full tours that have more than one meeting node) would make the score larger than the exact one, but the error is bounded and small [18]. Therefore, as an approximation algorithm, UISim is developed based on Eq. 4 in the following of the paper, and we will provide a deterministic multi-meeting tours correction method in Appendix, while the existing works [22], [26] rely on a probabilistic approximation of the induced error with random sampling.

The unification of computation space discussed above 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. 1) applies to any mode by enumerating and prioritizing the corresponding partial tours P_A and P_B, which we will discuss in Sect. 4.

4 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.

4.1 Hub-based benefit model

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 $p : x \rightarrow w_1 \rightarrow \cdots \rightarrow w_k$ with length $\mathcal{L}(p)$, is the probability of reaching x from w_k through p in a reverse 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, nodes with a large number of in-neighbors significantly decay the reachability of the tours passing through. In other words, the importance of partial tours can be indicated by the number of *high in-degree nodes*. On the other hand, the symmetry of two partial tours can also be indicated by the number of high in-degree nodes they pass through. Tours passing through more high in-degree nodes tend to be longer in terms of their natural length, and vice versa. While this correlation is intuitive, we also empirically verified it, and found that the average correlation coefficient of the number of high-degree nodes in a tour and the tour length in the real-world datasets is around 0.99.

In summary, the number of high in-degree nodes is effective to measure both tour importance and symmetry. We also refer to the high in-degree nodes as *hub nodes* and 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:

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

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.

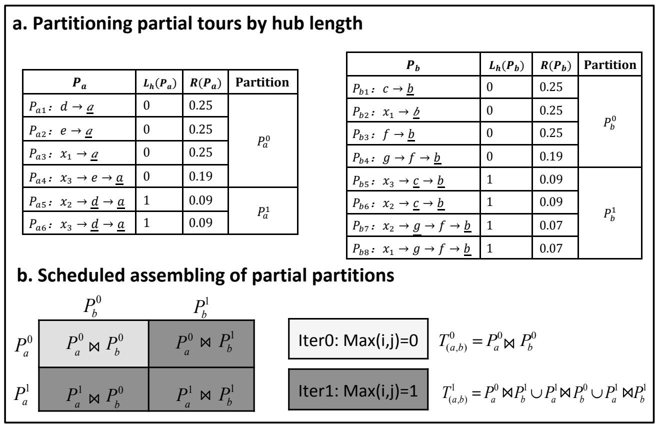
More formally, for any two assemblies $A_{ij} : P_u^i \bowtie P_v^j$ and $A_{i'j'} : P_u^{i'} \bowtie P_v^{j'}$, we should schedule A_{ij} in an earlier iteration than $A_{i'j'}$, denoted by $A_{ij} \prec A_{i'j'}$, with the following criteria:

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

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.

4.2 Prioritized SimRank approximation with benefit model

To leverage the benefit model for a prioritized SimRank approximation, We propose to integrate the above 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



a. Partitioning partial tours by hub length

P_a	$L_h(P_a)$	$R(P_a)$	Partition
$P_{a1}: d \rightarrow \underline{a}$	0	0.25	P_a^0
$P_{a2}: e \rightarrow \underline{a}$	0	0.25	
$P_{a3}: x_1 \rightarrow \underline{a}$	0	0.25	
$P_{a4}: x_3 \rightarrow e \rightarrow \underline{a}$	0	0.19	
$P_{a5}: x_2 \rightarrow \underline{d} \rightarrow \underline{a}$	1	0.09	P_a^1
$P_{a6}: x_3 \rightarrow \underline{d} \rightarrow \underline{a}$	1	0.09	

P_b	$L_h(P_b)$	$R(P_b)$	Partition
$P_{b1}: c \rightarrow \underline{b}$	0	0.25	P_b^0
$P_{b2}: x_1 \rightarrow \underline{b}$	0	0.25	
$P_{b3}: f \rightarrow \underline{b}$	0	0.25	
$P_{b4}: g \rightarrow f \rightarrow \underline{b}$	0	0.19	
$P_{b5}: x_3 \rightarrow \underline{c} \rightarrow \underline{b}$	1	0.09	P_b^1
$P_{b6}: x_2 \rightarrow \underline{c} \rightarrow \underline{b}$	1	0.09	
$P_{b7}: x_2 \rightarrow \underline{g} \rightarrow f \rightarrow \underline{b}$	1	0.07	
$P_{b8}: x_1 \rightarrow g \rightarrow f \rightarrow \underline{b}$	1	0.07	

b. Scheduled assembling of partial partitions

$$\begin{array}{c|c|c} & P_b^0 & P_b^1 \\ \hline P_a^0 & P_a^0 \bowtie P_b^0 & P_a^0 \bowtie P_b^1 \\ \hline P_a^1 & P_a^1 \bowtie P_b^0 & P_a^1 \bowtie P_b^1 \end{array}$$

$$T^0_{(a,b)} = P_a^0 \bowtie P_b^0$$

$$T^1_{(a,b)} = P_a^0 \bowtie P_b^1 \cup P_a^1 \bowtie P_b^0 \cup P_a^1 \bowtie P_b^1$$

Fig. 3: An example of prioritized generation of full tour partitions with benefit model.

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:

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

Example: Scheduled assembling of full tours. Fig. 3 illustrates the process of generating the full tours of $s(a, b)$ with partial tours in a prioritized manner. First, the partial tours in P_a and P_b are partitioned by their hub lengths into P_a^0, P_a^1 and P_b^0, P_b^1 respectively. As we can see, the reachabilities of tours in P_a^0 (or P_b^0) are smaller than that of tours in P_a^1 (or P_b^1). Next, according to their priority index $Max(i, j)$, tours in P_a^0 are assembled with tours in P_b^0 ($Max(0, 0) = 0$) to generate the most important full tours $T^0_{(a,b)}$ in iteration-0, and the other assemblies of partial tour partitions with $Max(i, j) = 1$ (*i.e.*, $P_a^0 \bowtie P_b^1$, $P_a^1 \bowtie P_b^0$, and $P_a^1 \bowtie P_b^1$) are scheduled to generate tours $T^1_{(a,b)}$ in iteration-1.

With the scheduled generation of full tour partitions, 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.

5 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 $G^i(u|*) = \{p : u \leftarrow\!\!\!\rightsquigarrow v \mid v \in V; \mathcal{L}_h(p) \le i\}$. 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.

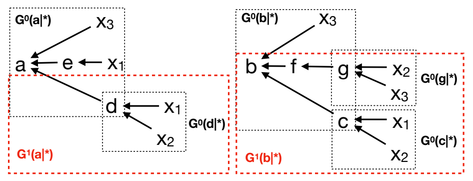
Next, to *assemble full tours* over partial-tour partitions, we notice that partial tours in a partition can have different natural length, while the valid query tours should have the same length on either side of the meeting nodes by definition (Eq. 4). Thus, to assemble two partial-tour sets, we can skip those "mismatching" 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:

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

where M is the maximal natural length in computation (*i.e.*, the number of iterations required for the fixed-point method to converge [6]), and in $r^{i,l}(u|x)$ we expand the superscript of hub length i to also denote natural length l as i, l.

Example: Efficient SimRank estimation with graph expansion. Fig. 4 shows an example of estimating $\hat{s}^{(1)}(a, b)$ on our toy graph (Fig. 1) with hub nodes $H = \{a, b, c, d, g\}$. First, in Fig. 4(a) the prime subgraphs of query nodes $G^0(a|*)$, $G^0(b|*)$ are expanded at their border hubs d, g, c. The corresponding prime subgraphs $G^0(d|*)$, $G^0(g|*)$, $G^0(c|*)$ are assembled to generate the hub-length-1 tours in the expanded graphs $G^1(a|*)$ and $G^1(b|*)$. Then, tours in $G^1(a|*)$ and $G^1(b|*)$ are assembled length-by-length at common meeting nodes. Since there is no length-3 tours from node a, we will not get a match for length-3 tours in $G^1(b|*)$ as Fig. 4(b) shows. Finally, $\hat{s}^{(1)}(a, b)$ is estimated by aggregating the

(a) Hub-by-hub graph expansion



(b) Length-by-Length tours assembling

L	Tours in G¹(a|*)	Tours in G¹(b|*)	Valid full tours
1	$a \leftarrow x_3$	$b \leftarrow x_3$	$t_1 : a \leftarrow x_3 \rightarrow b$
2	$a \leftarrow e \leftarrow x_1$ $a \leftarrow d \leftarrow x_1$ $a \leftarrow d \leftarrow x_2$	$b \leftarrow c \leftarrow x_1$ $b \leftarrow c \leftarrow x_2$	$t_2 : a \leftarrow e \leftarrow x_1 \rightarrow c \rightarrow b$ $t_3 : a \leftarrow d \leftarrow x_1 \rightarrow c \rightarrow b$ $t_4 : a \leftarrow d \leftarrow x_2 \rightarrow c \rightarrow b$
3	N/A	$b \leftarrow f \leftarrow g \leftarrow x_2$ $b \leftarrow f \leftarrow g \leftarrow x_3$	N/A

(c) SimRank estimation $\hat{S}^{(1)}(a,b)$

$$\begin{aligned}
\hat{S}^{(1)}(a,b) &= \sum_{i,j\in(0,1)} R(P_a^i \bowtie P_b^i) && \longrightarrow \text{incremental estimation}\\
&= \sum_{t\in\{t_1,\ldots,t_4\}} R(t) && \longrightarrow \text{aggregated reachability}\\
&= r^{0,1}(a\,|\,x_3)\cdot r^{0,1}(b\,|\,x_3) && \longrightarrow \text{over length-1 tours}\\
&\left.\begin{aligned}
&+r^{0,2}(a\,|\,x_1)\cdot \underline{r^{0,1}(b\,|\,c)r^{0,1}(c\,|\,x_1)}\\
&+\underline{r^{0,1}(a\,|\,d)r^{0,1}(d\,|\,x_1)}\cdot \underline{r^{0,1}(b\,|\,c)r^{0,1}(c\,|\,x_1)}\\
&+\underline{r^{0,1}(a\,|\,d)r^{0,1}(d\,|\,x_2)}\cdot \underline{r^{0,1}(b\,|\,c)r^{0,1}(c\,|\,x_2)}
\end{aligned}\right\} && \begin{gathered}\text{over length-2 tours}\\ \textbf{(shared hub segments)}\end{gathered}
\end{aligned}$$

Fig. 4: An example of efficiently estimating $\hat{s}^{(1)}(a,b)$ with graph expansion and length-aware assembling.

reachabilities of all valid full tours where each reachability can be efficiently obtained by reusing the reachability of shared hub segments, illustrated in Fig. 4(c).

Specification for other modes. Such *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 $G(v|*) = \{p : v \leftsquigarrow w \mid w \in V\}$, or *out-subgraph* of u, denoted by $G(*|u) = \{p : u \rightsquigarrow w \mid w \in V\}$. It is worth noting that, no matter the direction of expansion, the prime subgraphs are always bordered by hub nodes (*i.e.*, high in-degree nodes), as by definition (Eq. 6) the reachability of tours in both in-subgraphs and out-subgraphs are decayed by the in-degree (rather than out-degree) of nodes they pass through.

Now consider how to efficiently span a set of partial tours P_U. To assemble full tours, the valid partial tours in P_U should start from certain meeting nodes, and thus 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, given a parital-pair SimRank query $Q = (A, B)$, we should compare $|P_A|$, $|P_B|$ with $|X_A|$, $|X_B|$ respectively, and decide the direction of expansion accordingly.

Example: Mode-specific spanning of tours. Fig. 5 gives the example of mode-specific tours spanning and assembling using our toy graph G. For single-pair estimation $\hat{s}^0(a,b)$, the prime *in-subgraph* of a and b are spanned and then the partial tours are assembled at the common meeting nodes x_3. For single-source estimation $\hat{s}^0(a,*)$, first the prime *in-subgraph* $G^0(a|*)$ are spanned, then the prime *out-subgraphs* of meeting nodes x_1 and x_3 in $G^0(a|*)$ are spanned to generate the full tours. For all-pair estimation $\hat{s}^0(*,*)$, the out-subgraphs of all meeting nodes x_1, x_2 and x_3 in G are spanned, and matched tours in each subgraph are assembled as full tours.

Algorithm 1: Incremental & Unified SimRank approximation

```
Input: a graph $G$; number of hub nodes $H$; number of
       iteration $\eta$; 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
    $r^{(\eta)}(u|*) \leftarrow GraphExp(G, \eta)$;
    $X_u \leftarrow$ meeting nodes in $r^{(\eta)}(u|*)$;
    $r^{(\eta)}(v|*) \leftarrow GraphExp(G, \eta)$;
    $X_v \leftarrow$ meeting nodes in $r^{(\eta)}(v|*)$;
    foreach $x \in X_u \cap X_v$ 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
    $r^{(\eta)}(u|*) \leftarrow GraphExp(G, \eta)$;
    $X_u \leftarrow$ meeting nodes in $r^{(\eta)}(u|*)$;
    foreach $x \in X$ do
        $r^{(\eta)}(*|x) \leftarrow GraphExp(G, \eta)$;
        foreach $v \in V$ do
            foreach $l \le M$ 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
    $X \leftarrow$ meeting nodes in $G$;
    foreach $x \in X$ do
        $r^{(\eta)}(*|x) \leftarrow GraphExp(G, \eta)$;
        foreach $v \in V$ do
            foreach $l \le M$ 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 $\hat{S}^{(\eta)}(Q)$.
```

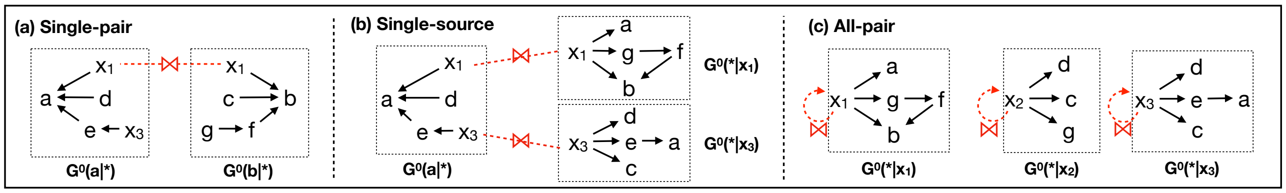


Fig. 5: An example of spanning partial tours in different modes.

Algorithm 2: GraphExp (Subroutine)

```
Input: a graph $G$; a root node $u$; type of subgraph $\kappa$, number of iterations $\eta$
Output: reachability over $\eta$-level subgraph $r_u^{(\eta)}$
1  if $\kappa$ = 'I' then
2      Construct prime in-subgraph $G^0(u|*)$ on $G$;
3      $r_u^0 \leftarrow r^0(u|*)$;
4  if $\kappa$ = 'O' then
5      Construct prime out-subgraph $G^0(*|u)$ on $G$;
6      $r_u^0 \leftarrow r^0(*|u)$;
7  $r_u^{(\eta)} \leftarrow r_u^0$;
8  if $\eta > 0$ then
9      for $i = 1 \ldots \eta$ do
10         $H_i \leftarrow$ hubs in $r_u^{i-1}$;
11         foreach $h_i \in H_i$ do
12             if $\kappa$ = 'I' then
13                 Expand $G^{i-1}(u|h_i)$ with $G^0(h_i|*)$;
14                 $r_u^i \leftarrow r^0(u|h_i)r^0(h_i|*)$;
15             if $\kappa$ = 'O' then
16                 Expand $G^{i-1}(h_i|u)$ with $G^0(*|h_i)$;
17                 $r_u^i \leftarrow r^0(*|h_i)r^0(h_i|u)$;
18         end
19         $r_u^{(\eta)} \leftarrow r_u^{(\eta)} + r_u^i$;
20     end
21 return $r_u^{(\eta)}$.
```

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, $|H|$ 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. 7. 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.

6 Complexity and error analysis

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

6.1 Complexity analysis

Time analysis. Since hub selection can be done in constant time, we focus the complexity analysis on 1) initial prime subgraphs processing cost, 2) prime subgraphs expansion cost, and 3) full tours assembling cost.

First, initial prime subgraph processing cost. Depending on the mode of SimRank query, we have different kinds of prime subgraphs—prime in-subgraphs of u and v for a single-pair query $Q = (u, v)$, prime in-subgraph of u and prime out-subgraphs of the corresponding meeting nodes $x \in X_u$ for a single-source query $Q = (u, V)$, and prime out-subgraphs of each meeting node $x \in X$ for an all-pair query $Q = (V, V)$. Given an average degree d and an input prime subgraph of size m (*i.e.*, of m nodes), while the processing time consists of the construction time and reachability computation time, the former is dominated by the latter. In particular, the reachabilities can be computed using the fixed point power-iteration method [31], which takes $O((m + md)I) = O(mdI)$ time, where I is the number of power iterations and md is the number of edges. Note that the number of power iterations I is the number of times to multiply the transition matrix in the power-iteration method until convergence, which is typically a small constant. In contrast, the construction of the prime subgraph is done using a depth-first search, which takes $O(m + md) = O(md)$ time only. Therefore, the overall processing time is $O(mdI)$.

Second, prime subgraph expansion cost. Prime subgraphs expansion is to extend the initial prime subgraphs at their border hubs (*i.e.*, assemble the prime subgraph 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,

TABLE 2: Time and Space Analysis of the Three Modes.

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

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 [6], 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. 7 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 Table 2. 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 $|X| \ll |V||\bar{X}|$.

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.

6.2 Error bound analysis

As UISim 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. Consider a random edge from the graph. Suppose the probability of the edge ending at any node is proportional to the node degree. Then, 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:

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

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 [15], 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 $0 \leq s(u, v) \leq 1$.

Remark. Our bound is built upon the skewed degree distribution of nodes. Moreover, we are assuming an undirected graph in the analysis here, which means the in- and out-degrees are the same. (In the case of a directed graph, the analysis should use in-degrees instead, *i.e.*, the probability of a directed edge ending at any node is proportional to the node in-degree.) In particular, hub nodes have higher degrees and partial tours are more likely to run into a hub node, *i.e.*, $\frac{d_H}{d_V} \gg \frac{|H|}{|V|}$. That means, the probability of a random edge ending at a hub node is skewed *w.r.t.* the degree, rather than uniformly distributed over all nodes.

7 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.

7.1 Experimental setup

Datasets. We use eight real-world datasets from different domains and with different properties and sizes summarized in Table 3. In particular, six datasets are used for baseline comparison where three smaller graphs of them are also used for parameter study,

TABLE 3: Summary of Datasets.

Dataset	Description	Directed	Nodes	Edges	Purpose
4Area	DBLP bibliographic network in four areas, similar to ref. [20], [3]	no	12 413	91 192	Parameter study, and comparison to baselines
WikiVote	Wikipedia administrator election network [10] (dangling nodes removed)	yes	1 300	39 456	
CondMat	Collaboration network of Arxiv Condensed Matter [10]	no	23 133	93 497	
enwiki2013	A snapshot of the English part of Wikipedia[10] [1]	yes	4 206 785	101 355 853	Comparison to baselines
it2014	A fairly large crawl of the .it domain [1]	yes	41 291 594	1 150 725 436	
Friendster	On-line gaming network [10]	no	65 608 366	1 806 067 135	
Gnutella	Gnutella peer to peer network with several snapshots [10]	yes	62 586	147 892	Scalability study
Dblp	Full DBLP bibliographic network with several snapshots, similar to ref. [3]	no	2 073 13	25 759 41	

and two evolving graphs with several snapshots are used to test the scalability of UISim.

Environment. We implement all methods in C++, and evaluate them on a Linux system with 3.5GHz CPU and 192GB RAM.

7.2 Experiments on smaller graphs

We first evaluate the algorithms on three smaller graphs, 4Area, WikiVote and CondMat. As the ground truth, the exact SimRank scores are computed by the power-iteration method with 55 iterations which ensures at most 10^{-12} absolute error.

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 *Absolute Error* (AbsErr) and *Relative Goodness* (RG). For a node pair, suppose its exact SimRank score is s and the estimated score is $\hat{s}$. Subsequently, AbsErr is simply defined as $|s - \hat{s}|$, and RG as $\min\{(s+\delta)/(\hat{s}+\delta), (\hat{s}+\delta)/(s+\delta)\}$ where δ is a small number 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 K nodes in the ranking, where $K = \{10, 20, 30\}$ on smaller graphs and $K = \{200, 300, 500\}$ on larger graphs. The average AbsErr (AvgErr) can be computed on the exact and estimated SimRank scores of these K 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 AvgErr 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 K lists. 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 $K = \{200, 500, 1000, 1500\}$ most similar pairs with largest SimRank scores. We also use AvgErr, RAG and Prec as our accuracy metrics. Note that, as analyzed in Sect. 1, returning the K most similar pairs from all-pair SimRank results actually solves the top-K SimRank Join problem, and thus we also compare all-pair UISim to top-K SimRank Join algorithm in our experiments.

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 UISim and discuss how to set the parameters. For single-source queries, we report the results on top K=20 results and for all-pair queries, top K=200 results. The results are shown in Fig. 6 and Fig. 7 respectively. Note that, for a consistent presentation, we plot the complement of AbsErr (or AvgErr) instead, *i.e.*, 1-AbsErr (or 1-AvgErr). Thus all the metrics indicate a better accuracy with a larger value.

Number of hubs. We first illustrate the effect of varying number of hubs $|H|$ in Fig. 6, 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. 6. 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. 6, most drops in accuracy are very minor while query processing becomes much faster, which is consistent with our theoretical analysis in Sect. 6. 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. 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. 7, 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 $\eta = 1$. It is interesting to observe that, the experiments also reveal the different contribution of SimRank increments $\hat{s}^k(u, v)$ in our scheduled approximation–when $\eta = 0$, only the most important increment $\hat{s}^0(u, v)$ contributes to the final scores, while when $\eta = 1$, the first two increments $\hat{s}^0(u, v)$ and $\hat{s}^1(u, v)$ make contributions, and so on. Thus, the results validate that the increments with small k contribute more

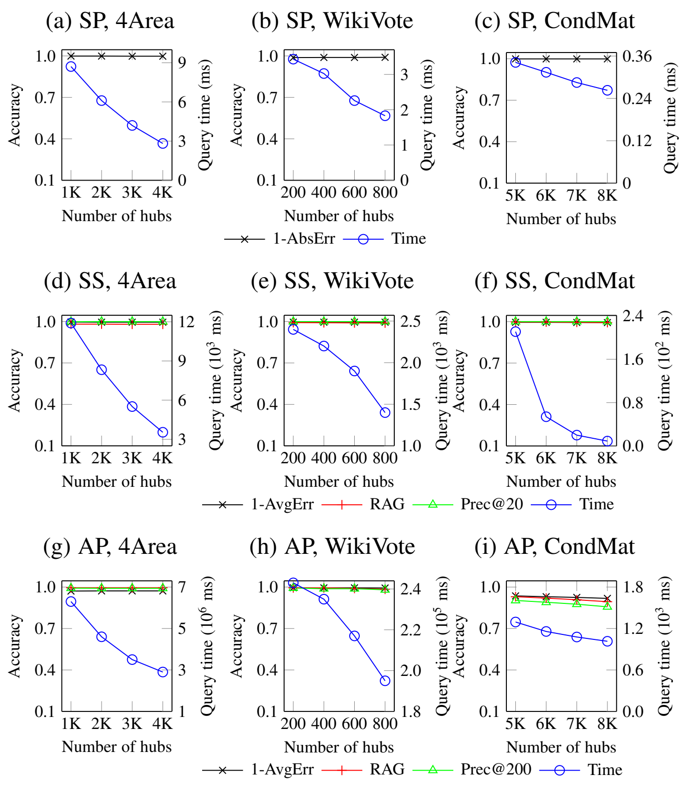


Fig. 6: Impact of number of hubs on accuracy metrics: AbsErr, AvgErr, RAG, Prec@K (left y-axis) and query time (right y-axis) in three modes: single pair (SP), single source (SS) and all pair (AP).

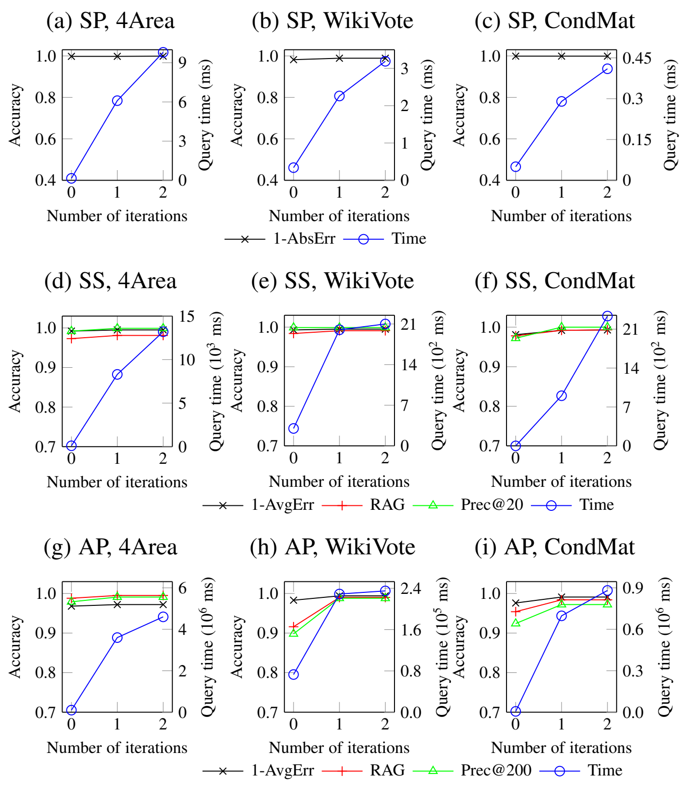


Fig. 7: Impact of number of iterations on accuracy metrics: AbsErr, AvgErr, RAG, Prec@K (left y-axis) and query time (right y-axis) in three modes: single pair (SP), single source (SS) and all pair (AP).

than the increments with large k.

Comparison to baselines. We compare UISim with the state-of-the-art competitors in each query mode: BLPMC [24], the single-pair solution; ProbeSim [14], PRSim [26] and SimPush [19], the single-source solutions; FLP [25], the all-pair solution, and TreeWand [21], the SimRank Join solution. Since PRSim is an index-based algorithm, for fair comparison, we also perform its index-free version by setting the number of precomputed hubs to zero. In the following, We refer to the original index-based PRSim as **PRSim w/**, and its index-free version as **PRSim w/o**. The threshold-based SimRank Join method [17] is not included as a baseline because it has a different setting where the threshold of similarity need to be specified. When the threshold is set to zero, it reduces to all-pair SimRank and it is two orders of magnitude slower than FLP algorithm (*i.e.*, the state-of-the-art all-pair baseline in our experiments) as reported in Reference [25].

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 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 $C = 0.75$. 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 and PRSim is varied from 0.001 to 0.2 at the step of 0.001. 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. 8, where x-axis is the running time and y-axis is the accuracy. Here we only present AbsErr 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 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. 8, 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. More concretely, we compare different methods under configurations that give similar accuracy. We exhibit the detailed accuracy and the running time of several “accuracy-moderated” configurations in Table 4 and 5. As observed, to achieve similar accuracy, UISim runs faster than the respective baselines in each mode. For example, as shown in Table 5, UISim outperforms the strongest index-free single-source

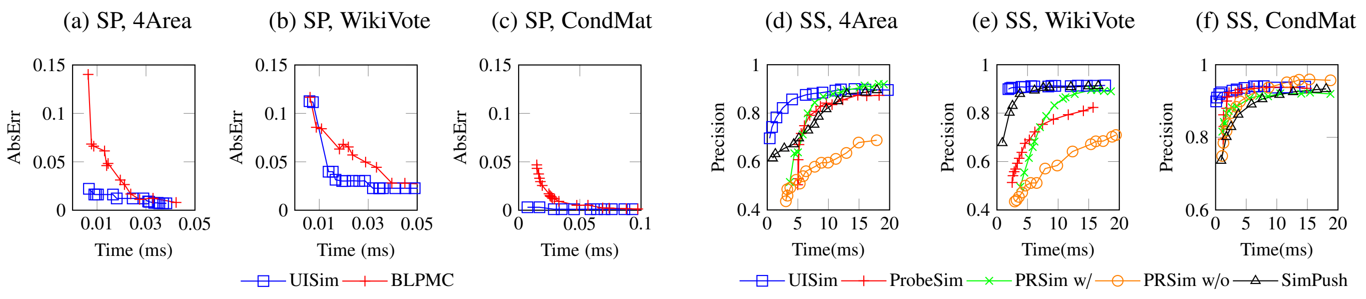


Fig. 8: Comparison of accuracy against time with baselines in single pair (SP) and single source (SS) modes.

TABLE 4: Detailed Comparison under Accuracy-moderated Configurations in Single-pair Mode.

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

baseline SimPush, and is even faster than the index-based PRSim w/ on smaller graphs. We also observe that SimPush is more sensitive to the graph structure. For example, SimPush is more inferior than UISim on 4Area (Fig. 8 (d)) and CondMat (Fig. 8 (f)) compared with their performance on WikiVote (Fig. 8 (e)). The reason is because generally the source graph of a query is also large (or dense) on 4Area/CondMat, and thus more time is wasted in identifying attention nodes and conducting reverse push at each level. On the other hand, UISim can achieve a good accuracy very fast while the baselines do not perform well within limited time. For example, as shown in Fig. 8(e), UISim can achieve a precision above 0.9 within 5ms, while given the same time, the best accuracy of the baselines is around 0.8. Such observation also validates the benefit of the "important-first" property of UISim discussed in Sect. 1.

For all-pair SimRank and SimRank Join, since the range of time we observed in UISim and the baseline TreeWand are vastly different (running one configuration can take several of hours for TreeWand), we only report the detailed results of nine configurations in Table 6. The results show that to achieve similar or better accuracy levels, UISim runs up to 4 times faster than FLP, and several orders of magnitude faster than TreeWand. It is worth noting that on 4Area and CondMat, the precision of all algorithms are around 1, but the AvgErr is relatively large. The reason is that there are many top ranked node pairs with the same exact SimRank scores, and the estimates for them are mostly the same (although not equal to the exact scores). Thus, the relative ranking obtained with approximate algorithms are still quite similar as the true ones.

We also evaluate the results with different K under the same configurations for single-source and all-pair queries. For single-source queries, we vary K from 10 to 30, and for all-pair queries, from 500 to 1500. Note that TreeWand is not included in the evaluation as it is running time increases with larger K (while other algorithms can compute the SimRank scores of all pairs, not just the K results). As shown in Fig. 9, UISim stably achieves higher accuracy than the baseline with larger K, while the accuracy of SimPush drops prominently with large K, mainly due to the pruning of non-attention nodes. For all-pair queries, since the precision of UISim and FLP are similar in some cases, we also evaluate the AvgErr with different K and observe that UISim can always obtain a smaller AvgErr of top K results.

7.3 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 (Table 3). We focus on the single-source node comparison with the baselines, 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 with Power Method. However, a latest work ExactSim [23] provides a probabilistic exact single-source Simrank solution that can achieve a precision of 7 decimal places with high probability. We thus use the results of ExactSim with $\varepsilon = 10^{-7}$ as ground truth on large graphs to evaluate UISim and the baselines ProbeSim, PRSim w/, PRSim w/o, and SimPush. We vary the parameters of different algorithms in the suggested range and compare them under "accuracy-moderated" configurations, similarly as we experiment with smaller graphs.

On each graph, 50 random queries are chosen and their average results are reported. We plot the trade-off between precision and query time (in log scale) in Fig. 8 (d), (e), (f), and observe that generally the accuracy of each method increases with more query time. However, the baselines need more time (by one order of magnitude) to achieve a higher precision above 0.9; while after that, the precision of all methods increase slightly, even with longer query time.

The detailed comparison of concrete settings with accuracy up to 0.97 are shown in Table. 7. For example, on Friendster, to achieve the precision around 0.93, UISim needs 10.75ms while the strongest baseline SimPush needs 105.7ms; to increase the query time (to 20.08ms for UISim and 128.5ms for SimPush), the precision only increase by 0.007 (K=500).

For thorough comparison, we also evaluate the accuracy of different algorithms with varied K. It can be observed that, UISim is up to 10 times faster than the strongest baseline SimPush while they produce results with a accuracy higher than 0.9.

TABLE 5: Detailed Comparison under Accuracy-moderated Configurations in Single-source Mode (K=20).

Dataset	Precision					AvgErr					RAG					Query Time(ms)					Index cost of PRSim w/	
	UISim	ProbeSim	PRSim w/o	PRSim w/	SimPush	UISim	ProbeSim	PRSim w/o	PRSim w/	SimPush	UISim	ProbeSim	PRSim w/o	PRSim w/	SimPush	UISim	ProbeSim	PRSim w/o	PRSim w/	SimPush	Space (KB)	Time (ms)
4Area	0.701	0.675	0.636	0.638	0.697	0.012	0.017	0.005	0.014	0.014	0.733	0.714	0.677	0.663	0.726	0.442	6.345	12.935	5.021	5.170	40	5.28
	0.821	0.799	0.818	0.799	0.817	0.011	0.012	0.002	0.009	0.011	0.846	0.822	0.840	0.821	0.838	2.530	7.656	44.729	6.610	9.652	56	5.52
	0.893	0.874	0.875	0.880	0.892	0.006	0.012	0.005	0.006	0.010	0.910	0.910	0.891	0.898	0.909	11.142	18.287	97.624	10.742	16.540	172	7.42
WikiVote	0.843	0.794	0.802	0.791	0.803	0.011	0.010	0.001	0.012	0.011	0.852	0.812	0.820	0.807	0.811	1.627	11.657	46.698	7.827	2.061	48	6.67
	0.883	0.862	0.855	0.843	0.882	0.010	0.010	0.001	0.011	0.009	0.892	0.875	0.869	0.856	0.890	4.191	22.342	102.699	10.043	4.890	68	8.30
	0.908	0.888	0.895	0.884	0.908	0.009	0.011	0.000	0.007	0.008	0.917	0.898	0.905	0.895	0.915	10.808	43.730	176.557	33.998	12.083	572	29.34
CondMat	0.905	0.905	0.867	0.855	0.897	0.017	0.022	0.007	0.016	0.008	0.944	0.942	0.945	0.917	0.949	0.230	1.819	2.001	2.093	6.703	152	9.88
	0.916	0.900	0.887	0.884	0.908	0.015	0.022	0.005	0.011	0.006	0.950	0.945	0.955	0.934	0.955	0.565	1.931	3.007	3.385	8.321	300	14.92
	0.924	0.923	0.914	0.926	0.925	0.012	0.022	0.003	0.007	0.004	0.954	0.953	0.969	0.959	0.964	1.988	2.684	7.146	6.487	12.669	612	24.87

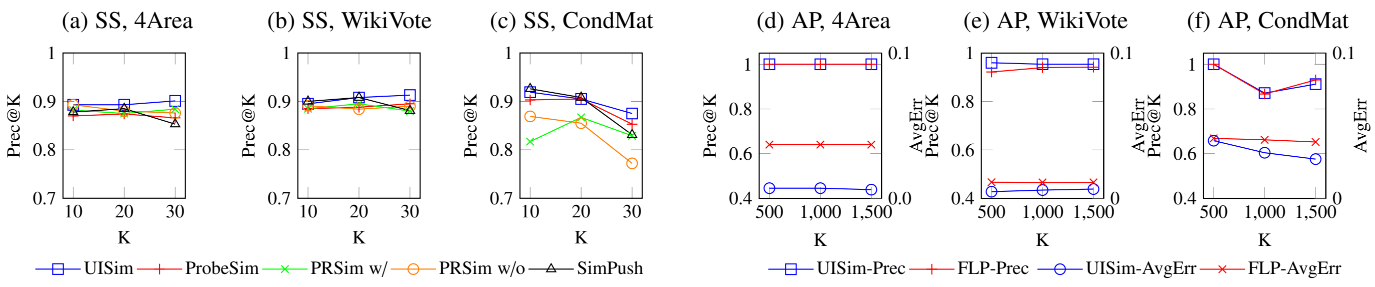


Fig. 9: Comparison of accuracy with different K under time-moderated configurations in single-source (SS) and all-pair (AP) modes.

TABLE 6: Detailed Comparison under Accuracy-moderated Configurations in All-pair Mode (K=200).

Dataset	Precision			AvgErr			**Time (s)**		
	UISim	FLP	TreeWand	UISim	FLP	TreeWand	UISim	FLP	TreeWand
4Area	1	1	1	.032	.034	.682	4.050	6.655	11740
	1	1	1	.030	.030	.680	5.089	8.782	11867
	1	1	1	.014	.015	.651	7.642	32.495	12045
WikiVote	.905	.930	.935	.011	.013	.0102	0.682	2.49	400.21
	.960	.950	.980	.004	.009	.0038	4.615	8.197	401.97
	.970	.955	.981	.003	.009	.0037	5.036	11.346	402.59
CondMat	1	1	.980	.044	.049	.547	1.291	1.509	2413.2
	1	1	.980	.043	.045	.546	1.457	1.750	2580.1
	1	1	1	.038	.040	.618	2.458	2.831	2696.1

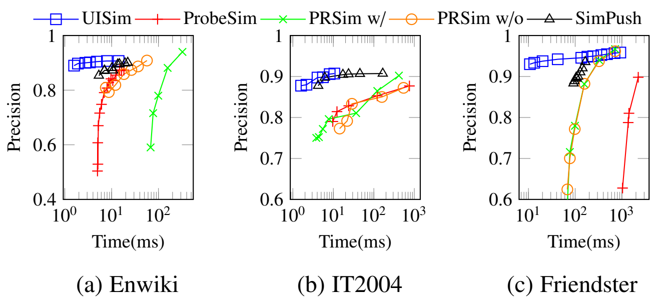


Fig. 10: Comparison of accuracy against time with baselines in single source mode on large graphs.

7.4 Scalability on growing graphs

Finally, we test the scalability of UISim on two growing graphs Gnutella and Dblp. 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 Table 8.

The key to scaling UISim is to increase the number of hubs $|H|$. As discussed in Sect. 7.2, using more hubs can reduce query time. Thus, we reasonably increase $|H|$ on a larger snapshot ($|H| = \beta \log(d)|V|$ with $\beta = 0.25$). As shown in Table 8, 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. On the other hand, with more hubs, only a sub-linear increase of memory cost is observed on growing graphs. The reason is that although more prime subgraphs of hubs are involved in the expansion when the number of hubs increases, the average size of prime subgraphs decreases as more tours would be truncated by hubs in the prime subgraphs.

8 Conclusion

In this paper, we presented an index-free approach to efficiently process all three modes of SimRank in a unified framework. 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.

Acknowledgment

We thank Yichen Shen (Zhejiang University City College), Zemin Liu (Singapore Management University) and Chengfeng Mao (University of Illinois at Urbana-Champaign) for their participation and contribution in this work. This material is based upon work supported by the Primary Research and Development Plan of Zhejiang Province, China (Grant No.2021C01164), the National Science Foundation IIS 16-19302 and IIS 16-33755, Zhejiang University ZJU Research 083650, Futurewei Technologies HF2017060011 and 094013, IBM-Illinois Center for Cognitive Computing Systems Research (C3SR)- a research collaboration as part of the IBM Cognitive Horizon Network, grants from eBay and Microsoft Azure, UIUC OVCR CCIL Planning Grant 434S34, UIUC CSBS Small Grant 434C8U, and UIUC New Frontiers Initiative. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

TABLE 7: Detailed comparison under accuracy-moderated Configurations in single-source mode on large graphs.

Dataset	K=500										K=300										Query Time(ms)					Index Cost of PRSim	
	Precision					AvgErr					Precision					AvgErr											
	UISim	SimPush	PRSim w/o	PRSim w/	ProbeSim	UISim	SimPush	PRSim w/o	PRSim w/	ProbeSim	UISim	SimPush	PRSim w/o	PRSim w/	ProbeSim	UISim	SimPush	PRSim w/o	PRSim w/	ProbeSim	UISim	SimPush	PRSim w/o	PRSim w/	ProbeSim	Space (MB)	Time (s)
	0.893	0.882	0.874	0.814	0.867	0.003	0.002	0.002	0.005	0.001	0.899	0.899	0.886	0.866	0.894	0.003	0.002	0.002	0.004	0.001	1.720	10.208	26.4	11	85.4	29.2	6.4
Enwiki2013	0.906	0.895	0.887	0.854	0.901	0.003	0.002	0.002	0.004	0.001	0.905	0.912	0.891	0.880	0.919	0.003	0.002	0.002	0.003	0.001	7.247	13.5	36.6	13.2	99	34.8	7.8
	0.907	0.900	0.908	0.897	0.936	0.004	0.001	0.002	0.003	0.001	0.906	0.917	0.917	0.900	0.945	0.004	0.002	0.002	0.003	0.001	14.11	22.06	56.8	17	160	45	9.6
	0.898	0.885	0.872	0.811	0.853	0.025	0.015	0.018	0.029	0.030	0.910	0.909	0.908	0.866	0.888	0.024	0.014	0.016	0.023	0.027	5.533	17.545	541	36.6	121.4	1475.9	61.5
IT2004	0.904	0.893	0.877	0.865	0.877	0.025	0.014	0.024	0.024	0.023	0.911	0.917	0.896	0.866	0.888	0.023	0.014	0.021	0.019	0.020	7.487	24.09	2450	121.8	757	3116.2	105.7
	0.907	0.917	0.890	0.902	0.880	0.024	0.011	0.024	0.016	0.024	0.912	0.927	0.912	0.921	0.918	0.023	0.012	0.020	0.015	0.020	8.838	165.15	14726	416	1385	6100.6	172
	0.930	0.898	0.882	0.881	0.898	2E-04	1E-04	2E-04	2E-04	1E-04	0.931	0.924	0.911	0.914	0.926	2E-04	1E-04	3E-04	3E-04	1E-04	10.75	105.7	158	157	2240	52.5	134.8
Friendster	0.937	0.910	0.937	0.940	0.955	2E-04	1E-04	2E-04	1E-04	1E-04	0.942	0.939	0.949	0.957	0.967	2E-04	1E-04	2E-04	2E-04	1E-04	20.08	128.5	323	325	4595	106.5	496.4
	0.942	0.936	0.961	0.967	0.971	2E-04	1E-04	1E-04	1E-04	1E-04	0.945	0.965	0.966	0.973	0.979	2E-04	1E-04	1E-04	1E-04	1E-04	42.33	164.4	717	716	6797	214.3	1263.2

TABLE 8: Scalability of UISim on Growing Graphs.

Dataset	# Nodes	# Edges	# Hubs	Query time (ms)		Memory (MB)	
				SP	SS	SP	SS
Gnutella1	6,301	20,777	816	0.36	200.8	3.69	4.10
Gnutella2	8,717	31,525	1,217	0.48	344.56	4.51	5.53
Gnutella3	10,876	39,994	1,538	0.52	412.2	5.02	6.14
Gnutella4	22,687	54,705	2,168	0.314	308.17	7.68	9.22
Gnutella5	36,682	88,328	3,500	0.323	275.7	11.57	13.31
Gnutella6	62,586	147,892	5,843	0.311	371.38	18.74	20.58
DBLP1	360,248	3,299,662	86,628	97.73	5709.6	158.7	172.03
DBLP2	588,076	5,942,059	147,681	374.83	10931	272.9	282.62
DBLP3	943,308	10,349,565	245,323	490.68	15905	457.3	481.28
DBLP4	1,585,596	18,862,938	426,294	2116.2	49780	785.4	796.7
DBLP5	2,073,139	25,759,412	567,163	2513.6	69744	1061	1113

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

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

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

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

[7] G. Jeh and J. Widom. Scaling personalized web search. In *WWW*, pages 271–279, 2003.

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

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

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

[11] 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.

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

[13] R. Li, X. Zhao, H. Shang, Y. Chen, and W. Xiao. Fast top-k similarity join for simrank. *Information Sciences*, 381:1–19, 2017.

[14] 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.

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

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

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

[18] 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.

[19] J. Shi, T. Jin, R. Yang, X. Xiao, and Y. Yang. Realtime index-free single source simrank processing on web-scale graphs. *arXiv preprint arXiv:2002.08082*, 2020.

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

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

[22] B. Tian and X. Xiao. Sling: A near-optimal index structure for simrank. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1859–1874, 2016.

[23] H. Wang, Z. Wei, Y. Yuan, X. Du, and J.-R. Wen. Exact single-source simrank computation on large graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 653–663, 2020.

[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] Z. Wei, X. He, X. Xiao, S. Wang, Y. Liu, X. Du, and J.-R. Wen. Prsim: Sublinear time simrank computation on large power-law graphs. In *Proceedings of the 2019 International Conference on Management of Data*, pages 1042–1059, 2019.

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

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

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

[30] W. Zheng, L. Zou, Y. Feng, L. Chen, and D. Zhao. Efficient simrank-based similarity join over large graphs. *Proceedings of the VLDB Endowment*, 6(7):493–504, 2013.

[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.

Appendix

In Sect. 3, 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 $u \leftsquigarrow x \leftarrow x_1 \rightarrow x \rightsquigarrow v$ 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* $u \leftsquigarrow x \leftarrow y \leftarrow x_2 \rightarrow z \rightarrow x \rightsquigarrow v$ 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 *3-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 $u \leftsquigarrow x \rightsquigarrow v$, 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 $t_x : u \leftsquigarrow x \leftarrow x_1 \rightarrow x \rightsquigarrow v$ is $R(t_x) = \frac{C}{|In(x)|^2} R(u \leftsquigarrow x \rightsquigarrow v)$. 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 given by $\sum_{x \in X} \frac{C}{|In(x)|} \hat{s}_x(u, v)$. □

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.

PLACE PHOTO HERE

Fanwei Zhu received her Ph.D. degree in Computer Science from Zhejiang University in 2012. She is currently an Associate Professor in Zhejiang University City College. Her research focuses on graph-based proximity search and social network analysis.

PLACE PHOTO HERE

Yuan Fang received his Ph.D. degree in Computer Science from University of Illinois at Urbana-Champaign in 2014. He is currently an Assistant Professor in the School of Information Systems, Singapore Management University. His research focuses on graph-based machine learning and data mining, as well as their applications for the Web and social media.

PLACE PHOTO HERE

Kai Zhang received his Bachalor's degree in Computer Science from Zhejiang University City College in 2020. He is currently a Research Assistant in Tsinghua University. His research focuses on information extraction and data mining.

PLACE PHOTO HERE

Hongtai Cao received his Bachelor of Engineering from Zhejiang University and Master of Science from the University of Southern California. He is a Ph.D. candidate in Computer Science at the University of Illinois Urbana-Champaign. His research interests include graph database systems and data analysis.

PLACE PHOTO HERE

Kevin Chen-Chuan Chang is a Professor in University of Illinois at Urbana-Champaign. His research addresses large-scale information access, for search, mining, and integration across structured and unstructured big data including Web data and social media. He also co-founded Cazoodle for deepening vertical data-aware search over the Web.

PLACE PHOTO HERE

Minghui Wu received his PhD degree in Computer Science and Engineering from Zhejiang University. He is a professor of Computer Science at Zhejiang University City College. His major interests include Artificial Intelligence, Big Data, Mobile Application, and Software Engineering.