Efficient SimRank-based Similarity Join Over Large Graphs *

Weiguo Zheng[1], Lei Zou[1†], Yansong Feng[1], Lei Chen[2] Dongyan Zhao[1]

[1]*Peking University, China;*
{zhengweiguo,zoulei,fengyansong,zhaody}@pku.edu.cn
[2] *Hong Kong University of Science and Technology, China;*
leichen@cse.ust.hk

ABSTRACT

Graphs have been widely used to model complex data in many real-world applications. Answering vertex join queries over large graphs is meaningful and interesting, which can benefit friend recommendation in social networks and link prediction, etc. In this paper, we adopt "SimRank" to evaluate the similarity of two vertices in a large graph because of its generality. Note that "SimRank" is purely structure dependent and it does not rely on the domain knowledge. Specifically, we define a *S*im*R*ank-based *j*oin (*SRJ*) query to find all the vertex pairs satisfying the threshold in a data graph G. In order to reduce the search space, we propose an estimated shortest-path distance based upper bound for SimRank scores to prune unpromising vertex pairs. In the verification, we propose a novel index, called *h-go cover*, to efficiently compute the SimRank score of a single vertex pair. Given a graph G, we only materialize the SimRank scores of a small proportion of vertex pairs (called *h-go covers*), based on which, the SimRank score of any vertex pair can be computed easily. In order to handle large graphs, we extend our technique to the partition-based framework. Thorough theoretical analysis and extensive experiments over both real and synthetic datasets confirm the efficiency and effectiveness of our solution.

1. INTRODUCTION

Recently, graph model has attracted extensive attentions in many fields, such as bioinformatics, chemistry, software engineering, traffic network and semantic web. Much real-world data in these domains can be modeled as graphs, where vertices represent different objects and edges model their pairwise relationships. Full studies over these graph data require effective graph data management techniques. Therefore, various types of queries, such as, subgraph search [32, 34], shortest-path query [3], reachability query [31, 30], pattern match query [4, 35, 7], and similarity join query [28] have been investigated. In our work, we focus on the similarity join problem between two vertex sets in a graph. Specifically, given two sets of vertices in a graph and a specified threshold θ, the results are the vertex pairs whose similarity scores are no less than θ.

In this paper, we would not adopt any domain dependent similarity function. Instead, we utilize "SimRank" [10], a structural-context similarity measure, to evaluate the similarity of two vertices in a large graph. SimRank is purely structure dependent which follows the intuition that "two objects are similar if they are related to similar objects"[10]. Because of its generality, SimRank has been widely used in many applications, such as link-prediction in social networks [20] and recommendation systems [1].

The *S*im*R*ank-based *j*oin (SRJ) query in this paper is defined as follows: Given two vertex sets U and V in a large graph G and a threshold θ, SRJ returns the vertex pairs (u, v) ($u \in U$ and $v \in V$), whose SimRank scores are no less than θ. The following examples demonstrate the usefulness of SRJ queries in real-life applications.

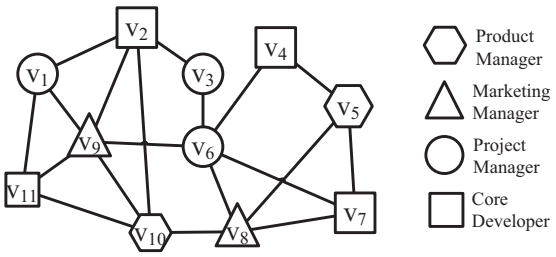


Figure 1: A Social Network

1) **IT Company Recruitment in Social Networks.** Assume that an IT company is launching a new project. It should recruit one project manager and one product manager. Recently, more and more employers prefer to use some professional social networks, such as LinkedIn [1], to seek candidates. Figure 1 shows a fictitious graph model (G) of a professional social network, where vertices represent active users and the edges indicate the friendship relations between two users. Assume that v_1, v_3 and v_6 are project managers, v_5 and v_{10} are product managers. As we know, as a successful group, the team members should have a healthy relationship

[1]http://www.linkedin.com/

[†]corresponding author: Lei Zou, zoulei@pku.edu.cn

*This work was supported by NSFC under Grant No.61003009, 61272344, 61202233. Yansong Feng and Dongyan Zhao were also supported by National High Technology Research and Development Program of China under Grant No. 2012AA011101. Lei Chen's work is supported in part by the Hong Kong RGC GRF Project No.611411, National Grand Fundamental Research 973 Program of China under Grant 2012-CB316200, HP IRP Project, Microsoft Research Asia Grant,and Grant from Huawei Noahs ark lab. Lei Zou's work was partially supported by State Key Laboratory of Software Engineering(SKLSE), Wuhan University, China.

Proceedings of the VLDB Endowment, Vol. 6, No. 7

among them and the low communication cost between group members is of great importance. Thus, we expect that the project manager and the product manager trust each other. In sociology, there is an interesting observation: the more similar two individuals are, the greater the trust between them is [23]. In Aristotle's Rhetoric and Nichomachean Ethics, Aristotle noted that people "love those who are like themselves" [2]. Plato observed in Phaedrus that "similarity begets friendship" [24]. Therefore, we would like to find a pair of project manager and product manager who are *similar* to each other. Here, we use *SimRank* to evaluate the similarity, which is based on the intuition that "two users are similar if they have many common friends". In this example, SRJ query will return some promising candidate pairs (in LinkedIn network), such as (v_1, v_{10}) and (v_5, v_6).

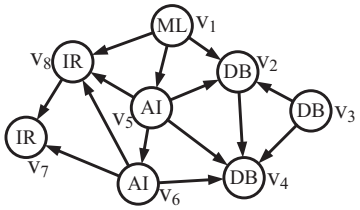


Figure 2: Scientific Paper Citation Network

2) **Bibliometric Analysis of Scientific Papers.** Usually, we model a bibliographic network as a directed graph, where vertices represent papers labeled with specified research areas (e.g., Database (DB), Information Retrieval (IR), Machine Learning (ML), Artificial Intelligence (AI)), a directed edge starting from v_i to v_j exists if paper v_i cites paper v_j. Figure 2 is such a model of bibliographic graph. Let us consider the following scenario: To enable cross-disciplinary studies in DB and IR, a scientist wants to find some paper pairs (v_i, v_j), where v_i and v_j are derived from DB and IR respectively, but they focus on a similar topic. Although there are some noteworthy similarity measures, such as co-citation [27] and bibliographic coupling [14], they only compute the similarity scores of any two vertices according to their immediate neighbors. In Figure 2, papers in DB area are not cited directly by any paper in IR area. Different from co-citation and bibliographic coupling, SimRank exploits the entire graph structure to determine the similarity rather than merely considering neighbors. Thus, SRJ query provides more cross-disciplinary work in DB and IR to users.

Although there exist some other pair-wise metrics for vertices, e.g. shortest path distances, SimRank captures the global structure of the graph to evaluate the structural-context "similarity" between two vertices. It is based on the random walk model, and it considers all possible paths between two vertices, while the shortest path distance emphasizes the "connection" and it only considers a single path (i.e., the shortest path) between two vertices. Due to their respective strengths, the two measures (SimRank and shortest path distances) are used in different applications. In this work, we focus on the "SimRank" measure.

*S*im*R*ank-based *j*oin (SRJ) queries have many potential applications, but it is not an easy task to answer the SRJ query, because it invokes expensive SimRank computation. A straightforward approach to answer SRJ queries is as follows: Given two sets of vertices U and V, we can first compute the pairwise SimRank scores for all the vertex pairs (u_i, v_j), where $u_i \in U$ and $v_j \in V$. Then, the vertex pairs satisfying the threshold are returned to users. Although lots of efforts have been made to compute SimRank in the literature [21, 8, 17, 18, 19], those methods are not efficient enough to answer SRJ queries. We briefly review the existing SimRank computation approaches and classify them into the following categories.

Computing All-Pairwise SimRank. Many methods focus on computing SimRank between all-pairwise vertices. Obviously, it is inefficient to employ these methods to compute SimRank scores between all-pairwise vertices on the fly to answer SRJ query, since we only need SimRank scores of a partial set of vertex pairs. On the other hand, it is not desirable to store SimRank scores between all-pairwise vertices for a large graph G, as it requires $O(|V(G)|^2)$ space complexity, where $|V(G)|$ is the number of vertices in graph G.

Computing Single-Source SimRank. Li et al. rewrite the SimRank equation into a non-iterative form, based on which it can derive approximate SimRank scores from one source vertex to all vertices [17]. However, its space cost is $O(k^2|V(G)|^2)$, where $k \ll |V(G)|$ [17].

Computing Single-Pair SimRank. Li et al. propose to compute SimRank of a single vertex-pair (v_i, v_j) by online enumerating all paths rooted at v_i and v_j, respectively [19]. Apparently, it is quite expensive to do that in a large graph.

Link-based Similarity Join. The most similar work to *SRJ* is the link-based similarity join proposed in [28], which delivers the top-k results. Our work differs from their study in that we focus on the threshold-based problem. Moreover, because the method in [28] adopts the iterative computation model to compute SimRank online, it is not efficient in a large graph.

In order to address SRJ queries, we adopt a filter-and-refine framework that is also considered in other graph database work as well. To avoid unnecessary SimRank computation, as far as we know, we are the first to propose a shortest-path distance-based upper bound for SimRank score. Due to the high cost of computing shortest-path distance on the fly, we adopt existing shortest-path distance estimation techniques to establish a new upper bound for $Sim(v_i, v_j)$ with low computation cost. The search space of SRJ queries can be reduced greatly with the upper bound. In the refinement phase, we need to compute SimRank scores of the remaining vertex pairs. Obviously, it is costly to compute $Sim(v_i, v_j)$ on the fly. However, it is not desirable to materialize all pairwise SimRank scores due to square space complexity $O(|V(G)|^2)$. Therefore, we propose to materialize SimRank scores of a partial set of vertex pairs (called *h-go cover* vertex pairs, see Definition 3.2) rather than those of all-pairwise vertex pairs in offline processing. Consequently, it reduces the storage cost greatly. Moreover, based on the SimRank scores of these h-go cover vertex pairs, we propose an efficient way to compute a single pair-wise SimRank score in online processing.

To summarize, we make the following contributions.

- In order to reduce the search space of SRJ queries, we propose a novel upper bound for SimRank scores.
- We propose an efficient way to compute the SimRank score of a vertex pair by materializing SimRank scores of a partial set of vertex pairs (called h-go cover vertex pairs). In order to handle large graphs, we propose a partition-based solution.
- We prove that finding the minimum number of h-go cover vertex pairs is NP-hard. Thus, two efficient algorithms are proposed to find the approximate solutions. More importantly, we provide thorough theoretical analysis about the effectiveness of our algorithms.
- We conduct extensive experiments over both real and synthetic graphs to confirm that the proposed method answers SRJ queries efficiently.

Organization. The rest of this paper is organized as follows. Section 2 defines the problem to be addressed in this paper and

Table 1: Frequently-used Notations

Notation	Definition and Description
G	A labeled unweighted simple graph
$N = \|V_G\|$	The number of vertices in G
v_a, v_b	Two vertices in G
$e(v_a, v_b)$	An edge in G
$d(v_a)$	the degree of v_a
$dist(v_a, v_b)$	the shortest distance between v_a and v_b
$Sim(v_a, v_b)$	The SimRank score of v_a and v_b
$R_k(v_a, v_b)$	The SimRank score of v_a and v_b on iteration k
G^b	The block-graph of G
$VC(G)$	The vertex cover of G
G^p	The vertex-pair graph of G
$\widetilde{G^p}$	The similarity graph of a vertex-pair graph G^p
X_{G^p}	The h-go cover of G^p
n	A node in G^p or $\widetilde{G^p}$
$S(n)$	The similarity score of node n in G^p or $\widetilde{G^p}$

gives an overview of our solution. To answer SRJ queries efficiently, Section 3 introduces an upper bound of SimRank score and presents two index techniques of computing *h-go* covers. Section 4 describes the operations for answering SRJ queries in large graphs. Section 5 reports the experimental results, followed by the related work in section 6. Finally, Section 7 concludes the paper.

2. PRELIMINARIES

In this section, we formally define the problem to be addressed and review the terminologies used in this paper. We also give an overview of our solution in this section. Table 1 lists the frequently-used notations throughout the paper. For clarity, we use the term "vertex" to denote the vertices in the original graph G. The vertices in other graphs, such as vertex-pair graph or similarity graph (defined in Definitions 3.1 and 3.5 respectively), are called "nodes".

2.1 Problem Definition

A labeled graph G is defined as $G = (V_G, E_G)$, where V_G is a set of vertices and E_G is a set of edges. In this paper, we focus on the undirected graph, although the method can be used in the directed graph with minor modifications.

DEFINITION 2.1. *(SimRank [10]) Given two vertices v_1 and v_2 in graph G, the SimRank score between v_1 and v_2 (denoted as $Sim(v_1, v_2)$) is defined as follows:*

$$Sim(v_1, v_2) = \begin{cases} 1, & \text{if} \quad v_1 = v_2; \\ \frac{c}{|I(v_1)||I(v_2)|} \sum_{i=1}^{|I(v_1)|} \sum_{j=1}^{|I(v_2)|} Sim(I_i(v_1), I_j(v_2)), & \text{otherwise} \end{cases} \quad (1)$$

where c is a decay factor between 0 and 1, $|I(v_1)|$ and $|I(v_2)|$ denote the number of neighbors of v_1 and v_2 respectively, $I_i(v_1)$ and $I_j(v_2)$ denote the i-th neighbor of v_1 and j-th neighbor of v_2, respectively.

Problem Definition (*SimR**ank-based **J**oin* (SRJ) Query)**[2]. Given two set of vertices U and V of a graph G (i.e., $U \subset V_G, V \subset V_G$) and a user specified threshold θ, SRJ delivers the vertex pairs (u, v) whose SimRank scores are no less than θ, i.e., $Sim(u, v) \geq \theta$, where $u \in U$ and $v \in V$.

[2]We only focus on the problem of threshold-based query in this paper. To answer the top-k query, we can sample the SimRank scores of the datasets, and then update the threshold dynamically based on the obtained answers. Further discussions of better methods to answer the top-k query are beyond the scope of this paper.

Figure 3 shows a running example. Suppose the two sets of vertices are $\{v_1, v_3, v_4\}$ and $\{v_2, v_5\}$ and $\theta = 0.1$. After computing the SimRank scores of G (suppose that the decay constant $c = 0.6$), we have $Sim(v_1, v_2) = 0.244$, $Sim(v_1, v_5) = 0.237$, $Sim(v_3, v_2) = 0.09$, $Sim(v_3, v_5) = 0.028$, $Sim(v_4, v_2) = 0.193$, and $Sim(v_4, v_5) = 0.066$. Three SimRank scores are larger than the threshold. Therefore, (v_1,v_2), (v_1,v_5) and (v_4,v_2) are the query results of SRJ.

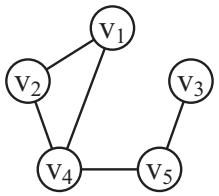


Figure 3: A Running Example Graph G

A baseline solution to solve this problem is as follows: Given two sets of vertices U and V, we perform a nested loop join algorithm. In the inner loop, for each candidate vertex pair (u, v), $u \in U$ and $v \in V$, we compute the SimRank score $Sim(u, v)$. And then the vertex pairs whose SimRank scores are no less than the threshold are returned as final results.

Obviously, it is an inefficient solution, since we need to compute SimRank scores for many vertex pairs. It is expensive to compute $Sim(\cdot, \cdot)$ on the fly in terms of query response time. To efficiently answer SRJ queries, another simple method is to materialize SimRank scores of all vertex pairs. Unfortunately, the space cost of storing all pairs of SimRank is extremely high, especially when the graph is large.

2.2 The Overview of Our Solution

In order to answer SRJ queries efficiently, we propose a filter-and-refine framework. Specifically, we first build an upper bound (for SimRank) with the light computation cost. Using the upper bound, we can filter out some vertex pairs whose SimRank upper bounds are less than the threshold θ. If a vertex pair (v_1, v_2) still survives after the pruning, we need to compute $Sim(v_1, v_2)$. We propose an efficient solution to compute $Sim(v_1, v_2)$ by materializing SimRank scores of a partial set of vertex pairs. The overall framework is given in Figure 4.

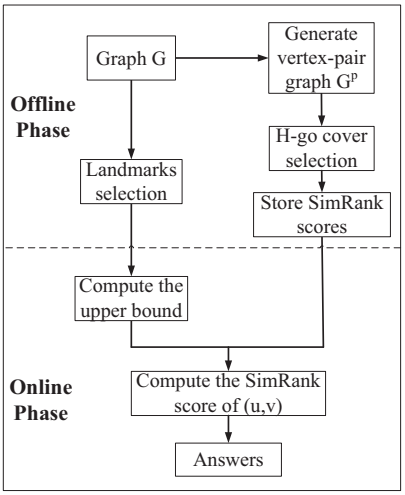


Figure 4: The Overview of Our Solution

Offline Phase. Since our proposed upper bound is based on the shortest path estimation technique, we select some landmark vertices in graph G and store the shortest distances between each vertex and these landmarks. We call them the *landmark index*.

Furthermore, given a graph G, we generate a vertex-pair graph G^p, over which a *h-go* cover is computed. We prove that minimizing the size of h-go cover is NP-hard. Thus, we design two heuristic solutions to select "h-go cover vertex pairs" for index construction. Finally, we materialize the *h-go* cover X together with corresponding SimRank scores as the index. We call them *h-go cover index*.

Online Phase. In the online phase, we first prune the search space to avoid the expensive SimRank computation, and then verify all the candidate pairs passing the filter.

1) *Pruning:* Since we design an estimated shortest path distance-based upper bound for SimRank score, based on the landmark index, we can filter out some vertex pairs whose upper bounds are smaller than θ. Therefore, many expensive SimRank computations could be avoided.

2) *Refining:* In the refining process, we need to compute SimRank scores for all candidate vertex pairs. Based on the h-go cover index, we design an efficient solution to compute the SimRank scores on the fly.

Furthermore, we also design a partition-based solution to handle SPJ queries over a very large graph, which is discussed in Section 4.

3. TECHNIQUES FOR SRJ QUERY

3.1 SimRank Upper Bound

Let us recall the SimRank definition in Equation 1. A solution to Equation (1) for a graph G can be reached by iterating to a fixed-point. For the *k-th* iteration, we introduce an iterative similarity function $R_k(v_1, v_2)$, which denotes the similarity score between v_1 and v_2 on the k-th iteration. Initially, $R_0(v_1, v_2)$ is defined as

$$R_0(v_1, v_2) = \begin{cases} 1, & \text{if} \quad v_1 = v_2, \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

Iteratively, $R_{k+1}(v_1, v_2)$ is computed from $R_k(\cdot, \cdot)$ as follows:

$$R_{k+1}(v_1, v_2) = \frac{c}{|I(v_1)||I(v_2)|} \sum_{i=1}^{|I(v_1)|} \sum_{j=1}^{|I(v_2)|} R_k(I_i(v_1), I_j(v_2)) \tag{3}$$

Theoretically, $Sim(v_1, v_2) = \lim_{k\to\infty} R_k(v_1, v_2)$ for all $v_1, v_2 \in V_G$. The time complexity required for k iterations is $O(kN^2d^2)$, where k is the number of iterations, N is the number of vertices of G, d is the average degree of vertices. Moreover, it requires $O(N^2)$ space to store the results $R_k(\cdot, \cdot)$. Hence, it is extremely time and space consuming to compute SimRank scores in large graphs.

According to the iterative computation model of SimRank, we give the following Lemma.

LEMMA 3.1. *For any two vertices v_1 and v_2, the difference between the (k+1)-th and the k-th iteration of SimRank scores holds:*

$$0 \le R_{k+1}(v_1, v_2) - R_k(v_1, v_2) \le c^{k+1}. \tag{4}$$

PROOF. 1) If $v_1 = v_2$ or $I(v_1) = \emptyset$ or $I(v_2) = \emptyset$, according to SimRank definition, $R_{k+1}(v_1, v_2) - R_k(v_1, v_2) = 0 < c^{k+1}$.

2) For the general case of $v_1 \neq v_2$, $I(v_1) \neq \emptyset$ and $I(v_2) \neq \emptyset$, the proof is organized by mathematical induction.

(*Induction Basis.*)($k = 0$)

$R_1(v_1, v_2) - R_0(v_1, v_2)) = R_1(v_1, v_2)$

$= \frac{c}{|I(v_1)||I(v_2)|} \sum_{i=1}^{|I(v_1)|} \sum_{j=1}^{|I(v_2)|} R_0(I_i(v_1), I_j(v_2))$

Since $0 \le R_0(I_i(v_1), I_j(v_2)) \le 1$,

$0 \le R_1(v_1, v_2) - R_0(v_1, v_2)) \le \frac{c \cdot |I(v_1)||I(v_2)| \cdot 1}{|I(v_1)||I(v_2)|} = c$,

the induction basis $R_1(v_1, v_2) - R_0(v_1, v_2)) \le c$ is proved.

(*Inductive step.*)

Provided that Equation (4) holds for a given integer $k(k > 0)$ for all vertex pairs, i.e.,

$$0 \le R_k(v_1, v_2) - R_{k-1}(v_a, v_b) \le c^k, \tag{5}$$

where $v_1, v_2 \in V(G)$

let us prove Equation (4) holds for $(k + 1)$ as well.

$R_{k+1}(v_1, v_2) - R_k(v_1, v_2)$

$= \frac{c}{|I(v_1)||I(v_2)|} \sum_{i=1}^{|I(v_1)|} \sum_{j=1}^{|I(v_2)|} R_k(I_i(v_1), I_j(v_2))$

$\quad - \frac{c}{|I(v_1)||I(v_2)|} \sum_{i=1}^{|I(v_1)|} \sum_{j=1}^{|I(v_2)|} R_{k-1}(I_i(v_1), I_j(v_2))$

$= \frac{c}{|I(v_1)||I(v_2)|} \times \sum_{i=1}^{|I(v_1)|} \sum_{j=1}^{|I(v_2)|} [R_k(I_i(v_1), I_j(v_2)) - R_{k-1}(I_i(v_1), I_j(v_2))]$

Based on Equation (5),

$0 \le R_{k+1}(v_1, v_2) - R_k(v_1, v_2)$

$\le \frac{c}{|I(v_1)||I(v_2)|} \cdot |I(v_1)||I(v_2)| \cdot c^k = c^{k+1}$

(*Conclusion.*) According to the above analysis, if Equation (4) holds for a given integer $k(k > 0)$ for all vertex pairs, Equation (4) will hold for $k + 1$. We have proved that Equation (4) holds for $k = 0$. Therefore, according to the induction method, we can conclude that Lemma 3.1 holds. □

As shown in Lemma 3.1, in the computation process of SimRank, the SimRank score difference between the *(k+1)-th* and *k-th* iteration decreases exponentially in terms of the number of iteration steps. Based on this lemma, we can deduce an upper bound for SimRank score of a vertex pair.

Considering any two vertices v_1 and v_2, if their pairwise shortest path distance[3] is not shorter than h, the SimRank score is 0 in the first $\lfloor 0.5(h-1) \rfloor$ iterations. Specifically, we obtain the following lemma.

LEMMA 3.2.

$$dist(v_1, v_2) \ge h \Rightarrow R_k(v_1, v_2) = 0, 0 \le k \le \lfloor 0.5(h-1) \rfloor$$

where $h \ge 1$, $dist(v_1, v_2)$ is the shortest path distance between v_1 and v_2, $R_k(v_1, v_2)$ denotes the iterative similarity score between v_1 and v_2 in the k-th iteration.

PROOF. (Proof by Induction) *(Induction Basis.)* When $h = 1$, we know $v_1 \neq v_2$, and $R_0(v_1, v_2) = 0$ based on the initial case.

(*Inductive step.*) provided that Lemma 3.2 holds for a given h and $h \ge 1$, i.e., if $dist(v_1, v_2) \ge h$, $R_k(v_1, v_2) = 0$, $0 \le k \le \lfloor 0.5(h-1) \rfloor$.

Let us prove that it also holds for $h + 1$. For any two vertices v_1 and v_2 with $dist(v_1, v_2) \ge h + 1$, let $k_0 = \lfloor 0.5(h-1) \rfloor$. When h is an odd number, $\lfloor 0.5h \rfloor = k_0$, $R_k(v_1, v_2) = 0$, $0 \le k \le \lfloor 0.5h \rfloor$ based on the hypothesis. When h is an even number, $\lfloor 0.5h \rfloor = k_0 + 1$.

$$R_{k_0+1}(v_1, v_2) = \frac{c}{|I(v_1)||I(v_2)|} \sum_{i=1}^{|I(v_1)|} \sum_{j=1}^{|I(v_2)|} R_{k_0}(I_i(v_1), I_j(v_2)).$$

Because $dist(v_1, v_2) \ge h + 1$, the shortest paths between their neighbors are not shorter than $h - 1$. Since h is an even number, $\lfloor 0.5(h-2) \rfloor = k_0$. According to the hypothesis, $R_{k_0}(I_i(v_1), I_j(v_2)) = 0$ in the first $\lfloor 0.5(h-1) \rfloor$ iterations. Hence, if $dist(v_1, v_2) \ge h + 1$, $R_k(v_1, v_2) = 0$, $0 \le k \le \lfloor 0.5(h) \rfloor$. □

[3]Here, the shortest path distance means the minimal number of hops between v_1 and v_2, i.e., each edge weight is 1.

It has shown that if $dist(v_1, v_2) \geq h$, $R_k(v_1, v_2)$ is equal to 0 when $0 \leq k \leq \lfloor 0.5(h-1) \rfloor$. On the basis of Lemmas 3.1 and 3.2, we give a SimRank upper bound of any vertex pair.

THEOREM 3.1.

$$dist(v_1, v_2) \geq h \Rightarrow Sim(v_1, v_2) \leq \frac{c^{\lfloor 0.5(h-1) \rfloor + 1}}{1-c}$$

where $h \geq 1$.

PROOF. Initially, $Sim(v, v) = 1$ and others are 0, $v \in V(G)$. Consider any two vertices v_1 and v_2 and $dist(v_1, v_2) \geq h$. For the simplicity of presentation, let t denote $(\lfloor 0.5(h-1) \rfloor)$. We know that $R(v_1, v_2) = 0$ in the first t iterations according to Lemma 3.2.

Since $R_{k+1}(v_1, v_2) - R_k(v_1, v_2) \leq c^{k+1}$,
$R_{t+1}(v_1, v_2) \leq R_t(v_1, v_2) + c^{t+1} = c^{t+1}$
and then
$R_{t+s}(v_a, v_b) \leq \sum_{i=1}^{s} c^{t+i} \leq \frac{c^{t+1}(1-c^s)}{1-c}$
when $s \to \infty$,
$s(v_1, v_2) = \lim_{s \to \infty} R_{t+s}(v_1, v_2) \leq \frac{c^{t+1}}{1-c} = \frac{c^{\lfloor 0.5(h-1) \rfloor + 1}}{1-c}$. □

If we know the shortest path distance between two vertices, their upper bound of SimRank will be calculated according to Theorem 3.1. However, it is also expensive to compute the shortest path between two vertices on the fly. Fortunately, the shortest path distance estimation techniques are well studied in the literature [25, 29]. Therefore, we can utilize existing methods to deduce the lower bound for the shortest path distance between two vertices. Consequently, we draw the following theorem about the upper bound for SimRank score, which is used in our paper.

THEOREM 3.2.

$$dist_{LB}(v_1, v_2) \geq h \Rightarrow Sim(v_1, v_2) \leq Sim^u(v_1, v_2) \tag{6}$$

where $Sim^u(v_1, v_2) = \frac{c^{\lfloor 0.5(h-1) \rfloor + 1}}{1-c}$, $h \geq 1$ *and* $dist_{LB}(v_1, v_2)$ *denotes the lower bound for the shortest path distance between* v_1 *and* v_2 *in* G.

PROOF. Omitted due to space limit. □

To make our paper self-contained, we review the shortest path distance estimation technique in brief. Note that, any estimation method [25, 29] can be applied in our problem, since these methods are orthogonal to our solution. In this paper, we adopt the method in [25]. The main idea of [25] is to select a set W of landmark vertices and compute the approximation according to Equation (7).

$$dist_{LB}(v_1, v_2) = \max_{v_u \in W} dist_l(v_1, v_2) \tag{7}$$

where $dist_l(v_1, v_2)$ is defined in Equation (8).

$$dist_l(v_1, v_2) = |dist(v_1, v_u) - dist(v_2, v_u)| \tag{8}$$

Equation (8) is based on the triangle inequality. The key step of estimating the lower bound is how to find the landmarks. Here, we adopt the heuristic method as discussed in [25]: First, the vertex with the smallest degree is selected as the first landmark. Then, we select the next landmark which is the farthest away from all selected landmarks so far. The above process is iterated until that a predefined number of landmarks are selected. According to recent studies, shortest path distance estimation technique could provide precise distance estimation in a very large graph, even in a billion-node graph. Interested readers may refer to [25, 29] for details.

In summary, we select some landmark vertices in G and record the shortest path distance between each vertex and the landmark vertices. We call these as the *landmark index*, which can be used to compute the SimRank score upper bound (see Equations 6 and 7).

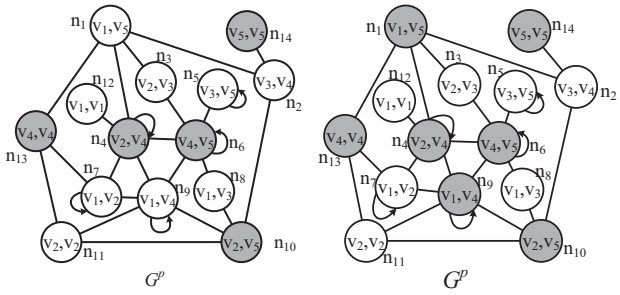


Figure 5: The 2-go cover and vertex cover of vertex-pair graph G^p

3.2 Index-based SimRank Computation

After the pruning process, there are some remaining vertex pairs whose similarity upper bounds are larger than θ. Now, we discuss how to compute SimRank scores for those remaining vertex pairs. As discussed in Section 3.1, the time complexity of computing SimRank is $O(kN^2d^2)$. Hence, it is time consuming to compute SimRank scores in an iterative model. It is not desirable to store SimRank scores of all vertex pairs, due to the large space complexity $O(N^2)$. Therefore, we propose a novel index-based solution to compute SimRank scores for all the remaining vertex pairs.

According to Equation (1), the SimRank score between two vertices is computed by the SimRank scores of their neighbors. Logically, we can construct a vertex-pair graph G^p, in which each node represents a vertex pair of G.

DEFINITION 3.1. *(Vertex-pair Graph)[10]. Given a graph G, the corresponding vertex-pair graph is defined as three tuples* $G^p = (V_p, E_p, S_p)$, *where* V_p *is a set of nodes,* E_p *is a set of edges,* S_p *is a set of SimRank scores, and*

- *Each node* $n \in V_p$ *represents a vertex pair* (v_i, v_j), *where* $v_i, v_j \in V(G)$. *The node n is associated with SimRank score* $Sim(v_i, v_j)$ *computed from the original graph G;*
- *There is an edge between nodes* $n_1(v_{i_1}, v_{j_1})$ *and* $n_2(v_{i_2}, v_{j_2})$ *in* G^p *if and only if: 1) both edges* (v_{i_1}, v_{i_2}) *and* (v_{j_1}, v_{j_2}) *exist in graph G; or 2) both edges* (v_{i_1}, v_{j_2}) *and* (v_{j_1}, v_{i_2}) *exist in graph G.*

Note that, each node n in G^p corresponds to a vertex pair in G. Thus, we use "node" and "vertex pair" interchangeably when the context is clear. Given the graph G in Figure 3, its corresponding vertex-pair graph G^p is shown in Figure 5(a). Each node in G^p denotes a vertex pair in G. There is an edge between $n_1(v_1, v_4)$ and $n_2(v_2, v_5)$ in G^p because $(v_1, v_2) \in E(G)$ and $(v_4, v_5) \in E(G)$. Note that, in this section, we assume that G^p can be cached in memory. We will discuss a partition-based solution to handle a large graph G in Section 4. Furthermore, G^P is not stored as index structures. Only *h-go cover vertex pairs* (Definition 3.2) together with their corresponding SimRank scores are index elements.

LEMMA 3.3. *In the vertex-pair graph* G^p, *the SimRank score of a node* n_k *can be computed through the SimRank scores of its neighbor nodes in* G^p *as*

$$S(n_k) = \frac{c}{|I(n_k)|} \sum_{i=1}^{|I(n_k)|} S(n_i) \tag{9}$$

where $I(n_k)$ is the neighbors of node n_k in G^p, and $n_i \in I(n_k)$.

PROOF. For each neighbor $n_i = (v_{ai}, v_{bi})$ of node $n_k = (v_{ak}, v_{bk})$, vertices v_{ai} and v_{bi} are the neighbors of v_{ak} and v_{bk} respectively.

$$\sum_{i=1}^{|I(n_k)|} S(n_i) = \sum_{i=1}^{|I(n_k)|} Sim(v_{ai}, v_{bi}) = \sum_{i=1}^{|I(v_{ak})|} \sum_{j=1}^{|I(v_{bk})|} Sim(I_i(v_{ak}), I_j(v_{bk})),$$

and $|I(n_k)| = |I(v_{ak})||I(v_{bk})|$. Therefore Equation (1) is equivalent to Equation (9). □

According to Lemma 3.3, given a node in G^p, its SimRank score can be calculated from its neighbors in G^p. Therefore, we propose to store the SimRank scores of a set of nodes in G^p, based on which, the SimRank score of any node can be recovered easily. To achieve that, we propose "h-go cover vertex pairs" and "path-tree" as follows.

DEFINITION 3.2. *(*h-go cover*). A h-go cover of a vertex-pair graph G^p, denoted as X_{G^p}, is a set of vertices whose removal leaves a graph without simple paths longer than h.*

According to the *h*-go cover definition, for each path of length no shorter than h in G^p, at least one of its nodes is contained in X_{G^p}. For notation simplicity, we also use X instead of X_{G^p} when the context is clear.

DEFINITION 3.3. *(*path tree*). Given a node n in an vertex-pair graph G^P, the path tree of node n is defined as a tree $T(n)$, where*

- *n is the root and its neighbors in G^p are its children in $T(n)$.*
- *The children of an intermediate node in $T(n)$ are its neighbors in G^p except for its ancestors in $T(n)$.*

THEOREM 3.3. *Given a path $T(n)$ and a h-go cover X of G^p, $T(n)$ must be* besieged *by X and the* besieged depth *is h, namely, for each branch P (in $T(n)$) with length longer than h, there must exist at least one node n', where $n' \in P \wedge n' \in X$, and the number of hops from n to n' is no larger than h.*

PROOF. It can be proved according to Definitions 3.2 and 3.3. □

A path tree of node $n_2(v_3, v_4)$ is shown in Figure 6, and this path tree is besieged by the grey nodes. The part (of the path tree) that is besieged by the grey nodes is called *besieged region*. Note that, the same vertex pair may occur twice or more times in a path tree. For example, (v_1, v_4) occurs three times in the path tree, as shown in Figure 6.

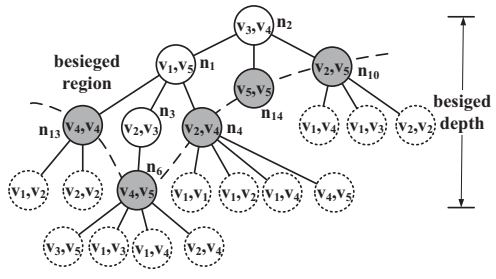


Figure 6: An example of path tree

If we only store the SimRank scores of nodes in X_{G^p}, for each node $n \notin X_{G^p}$, the SimRank score of n could be calculated from the SimRank scores of nodes in X_{G^p}. Let us first illustrate the main idea by the following running example. The grey nodes X=$\{n_4(v_2, v_4)$, $n_6(v_4, v_5)$, $n_{10}(v_2, v_5)$, $n_{13}(v_4, v_4)$, $n_{14}(v_5, v_5)\}$ in Figure 5(a) are a *2-go cover* of G^p. Assume that we need to compute the SimRank score of $n_2(v_3, v_4)$. According to Definition 3.3, we generate the path tree of node n_2, i.e., $T(n_2)$. Note that, it is not necessary to generate the whole path tree $T(n_2)$. Instead, we only consider the region besieged by *h*-go cover X. According to SimRank definition (Equation 1), we can get a system of linear equations in three variables ($S(n_1)$, $S(n_2)$ and $S(n_3)$) as follows:

$$\begin{cases} S(n_2) = \frac{c}{3}(S(n_1) + \underline{S(n_{10})} + \underline{S(n_{14})}) \\ S(n_1) = \frac{c}{4}(S(n_2) + S(n_3) + \underline{S(n_4)} + \underline{S(n_{13})}) \\ S(n_3) = \frac{c}{2}(S(n_1) + \underline{S(n_6)}) \end{cases}$$

where $S(n_4)$, $S(n_6)$, $S(n_{10})$, $S(n_{13})$ and $S(n_{14})$ are known quantities, as n_4, n_6, n_{10} and n_{14} are 2-go cover nodes. Solving the system of linear equations leads to the SimRank score of n_2, i.e., $Sim(v_3, v_4)$, by existing algorithms, such as Gaussian elimination[9].

Obviously, different *h-go covers* in the vertex-pair graph G^p will affect the query performance. We postpone the discussion about selecting *h-go cover* to Section 3.3. Given a graph G, assume that its *h-go cover* vertex pairs X are given. We store the *h-go cover* vertex pairs together with their corresponding SimRank scores in a triple column table B (as shown in Table 2), where the vertex pair is the primary key. On the observation that the $Sim(v_x, v_x) = 1$, it is not necessary to store them.

Table 2: Table T_B with SimRank scores of 2-go cover vertex pairs

Vertex ID	Vertex ID	SimRank Scores
v_2	v_4	0.193
v_4	v_5	0.066
v_2	v_5	0.237
v_4	v_4	1
v_5	v_5	1

Given any two vertices v_1 and v_2 in G, we propose Algorithm 1 to compute $Sim(v_1, v_2)$ resorting to the *h*-go cover vertex pairs X (i.e., the table T_B) and the original graph G. Note that, the vertex-pair graph G^p is not stored as the index structure. If $Sim(v_1, v_2)$ is not stored in table T_B, according to Definitions 3.1 and 3.3, we generate the path tree T of $n(v_1, v_2)$ by depth-first-search (DFS) traversal over G from vertices v_1 and v_2, simultaneously. Once some branch of T encounters some vertex pair in *h*-go cover X, the branch is terminated. In this way, we can get the besieged region of T by *h*-go cover X (steps 4-13 in Algorithm 1). For each non-leaf node in T, we gain an equation according to SimRank definition. By going over all the non-leaf nodes in T, we will obtain a system of linear equations. Solving the system of linear equations leads to the SimRank score of $Sim(v_1, v_2)$.

Correctness. Each equation in the system of equations is generated according to Equation (9). So the analytic solutions are correct.

3.3 h-go Cover Selection

Given a path tree $T(n)$, the lower the besieged depth of $T(n)$ by *h*-go cover X is, the less variables the system of equations has. As a result, Algorithm 1 will have a faster online performance. Therefore, the key is how to select an optimal *h*-go cover in G^p. However, Theorem 3.4 tells us that it is a NP-hard problem.

THEOREM 3.4. *The problem of finding a h-go cover that has the fewest nodes is NP-hard.*

Algorithm 1 *QuerySim*(v_1, v_2, T_B, G)

Require: Two vertices v_i, v_j in G and the table T_B with SimRank scores of h-go cover vertex pairs.
Ensure: $Sim(v_1, v_2)$.

```
1: if Sim(v1, v2) is stored in T_B then
2:     return Sim(v1, v2);
3: else
4:     push the vertex pair (v1, v2) into stack st
5:     while st ≠ φ do
6:         pop the top vertex pair (vx, vy)
7:         add (vx, vy) into tree T
8:         for each neighbor v'x of vx do
9:             for each neighbor v'y of vy do
10:                if vertex pair (v'x, v'y) ∉ T_B then
11:                    push (v'x, v'y) into stack st
12:                else
13:                    add (v'x, v'y) into tree T
14:    for each non-leaf vertex pair (vx, vy) ∈ T do
15:        generate an equation based on Equation (9)
16:    Sim(v1, v2) ← solve the system of linear equations
17:    return Sim(v1, v2)
```

PROOF. We can reduce the set cover problem which is known to be NP-hard to a *h-go cover* problem. Given an instance of the set cover problem, $U = \{u_1, u_2, \dots u_k\}$ is the universe elements, n sets $S = \{s_1, s_2, \dots s_n\}$ where s_i $(1 \le i \le n)$ is a set. We can replace each element u_i with a h-length path p_j in G, i.e., there is a bijection $f(u_i) \leftrightarrow p_j$ for all the h-length paths. What is more, the complexity of enumerating all the *h-length* paths in G is polynomial. The set cover problem is reduced to *h-go cover* problem. Since the optimization version of the set cover is NP-hard, finding a *h-go cover* that uses the fewest nodes is NP-hard. □

Therefore, we propose two heuristic methods to select h-go covers in a vertex-pair graph G^p.

3.3.1 Set Cover based Method

We have proved that finding a h-go cover with the minimum number of nodes in G^p is a NP-hard problem by reducing the minimum set-cover (MSC) problem to our case. Therefore, we adopt the greedy algorithm of MSC to find the h-go cover. Specifically, for each node $n \in G^p$, we enumerate all paths with lengths no longer than h containing node n. All paths are collected to form a path-set PS. We select a node n that appears in the maximum number of paths in P. The node n is put into X and all paths containing n are removed from PS. We repeat the above steps until no path is left in PS. The details are described in Algorithm 2.

THEOREM 3.5. *The approximate ratio of the set-cover based method for finding h-go cover is* $2 \times h \ln(|V(G)| \times d_p)$*, where* $|V(G)|$ *denotes the number of vertices in* $V(G)$ *and* d_p *denotes the average vertex degree in* G^p.

PROOF. It can be proved according to the set-cover problem. □

Although the set-cover based method provides a good approximate ratio to select go cover X, it requires to enumerate all length-h paths in G^p. It may be inefficient in terms of time and space complexity. Therefore, we propose the second method to select h-go cover X.

3.3.2 Similarity Graph based Method

DEFINITION 3.4. *(Vertex Cover). A* vertex cover *over a given graph G is a set of vertices such that each edge of G is adjacent to at least one vertex of this set.*

Algorithm 2 *XSelection*(G^p, h)

Require: G^p and a given constant h.
Ensure: *h-go cover* X.

```
1: a set PS ← φ
2: for each n_i ∈ G^p do
3:     compute the h-length paths that contains n_i
4:     number these paths with a unique id
5:     add id into PS
6:     construct a path set s_i consisting of these ids.
7: h-go cover X ← φ
8: while PS ≠ φ do
9:     select the set s with maximum size
10:    i ← the id of s
11:    X ← n_i
12:    for each element e in s do
13:        remove e from PS
14: return X
```

We denote a vertex cover of a graph G^p as $VC(G^p)$. Finding a vertex cover with the minimum number of vertices is a classical NP-hard problem. Lots of approximate algorithms have been proposed to find the approximate vertex cover [6, 16, 12]. In this paper, we adopt a simple strategy to compute the vertex cover. Specifically, repeatedly take both endpoints of an edge into the vertex cover, and then remove them together with all the vertices incident to them, until that all all edges are covered [5].

According to the definition of vertex cover, it is straightforward to know that if a node is not in $VC(G^p)$, all its neighbors must be in $VC(G^p)$. An example of vertex cover in G^p is the set of grey vertices in Figure 5(b).

Clearly, $VC(G^p)$ is a *1-go cover* of G^p. Nevertheless, it only saves about half of the storage cost. Obviously, when the graph is large, it is still costly to store the index. To further reduce the storage cost, we propose the *similarity graph* to select a h-go cover X in G^p.

DEFINITION 3.5. *(Similarity Graph). Given a vertex-pair graph* $G^p = (V_p, E_p, S_p)$*, the similarity graph of* G^p*, denoted by* $\widetilde{G^p}^1 = (V_s, E_s, S_s)$*, is defined as follows:*

- $V_s = VC(G^p)$;
- *We introduce an edge* (n_i, n_j) *into* $\widetilde{G^p}^1$ *if and only if at least one of following conditions hold:*
  1) $n_i \in I(n_j)$*, where* $I(n_j)$ *denotes the neighbor nodes of* n_j *in* G^p*; or*
  2) $\exists n \in V(G^p), n \notin VC(G^p) \wedge n \in I(n_i) \wedge n \in I(n_j)$.
- S_s *is the SimRank score set of* V_s*, and the score of each node is equal to that of the corresponding node in* G^p.

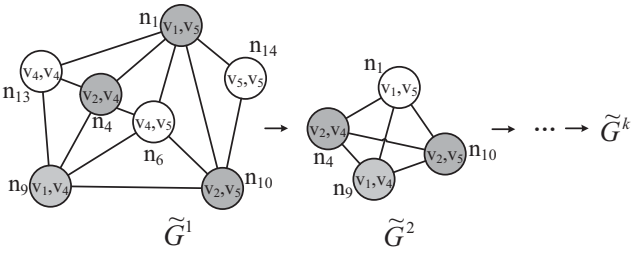


Figure 7: The process of generating similarity graphs

Analogously, given a similarity graph $\widetilde{G^p}^1$, we can also define the second level similarity graph $\widetilde{G^p}^2$ from the first level similarity graph $\widetilde{G^p}^1$ based on Definition 3.5. Iteratively, we can define

the k-th level similarity graph, as shown in Figure 7. We repeat the above process until that the vertex cover over $\widetilde{G^p}^k$ is small enough (namely, $VC(\widetilde{G^p}^k)$ can be cached in memory). The algorithm of generating similarity graphs is presented in Algorithm 3. Note that, *similarity graphs* are not necessary to be stored as the index. We only store the vertex cover of the last level similarity graph as the h-go cover, namely $X = VC(\widetilde{G^p}^k)$. Theorem 3.6 gives the upper bound for besieged depth of each path tree, if we select $X = VC(\widetilde{G^p}^k)$.

Algorithm 3 *SG-generation*(G^p, k)

Require: A vertex-pair graph G^p, the level depth of the similarity graphs k.
Ensure: The h-go cover X of G^p.
1: Find the vertex cover $VC(G^p)$ for G^p
2: $VC(\widetilde{G^p}^0) \leftarrow VC(G^p)$
3: $i \leftarrow 1$
4: **for** $i \leq k$ **do**
5: Generate the similarity graph $\widetilde{G^p}^i$ for $VC(\widetilde{G^p}^{i-1})$ based on Definition 3.5
6: $VC(\widetilde{G^p}^i) \leftarrow$ Find vertex cover for $\widetilde{G^p}^i$
7: $i \leftarrow i + 1$
8: $X \leftarrow$ Find the vertex cover for $\widetilde{G^p}^k$
9: **return** X

THEOREM 3.6. *Given a vertex-pair graph G^p, the vertex pairs selected by Algorithm 3 is a h-go cover, where $h = 2^{k+1} - 1$, where k denotes the level of the similarity graph.*

PROOF. The proof is organized by mathematical induction.

Induction Basis: When we only generate the similarity graph for G^p, i.e., k =0, the distance between each uncovered vertex and that in the cover set is 1, therefore $h = 2^{0+1} - 1 = 1$.

Inductive step: provided that it holds for $\widetilde{G}^k$, i.e., $h = 2^{k+1} - 1$.

The distance of two nodes in $\widetilde{G}^k$ is at most 2^k according to the definition of similarity graph. When we generate the $\widetilde{G}^{k+1}$ from $\widetilde{G}^k$,

$$h = 2^{k+1} - 1 + 2^{k+1} = 2^{k+2} - 1. \quad \square$$

Notice that, Theorem 3.6 gives the upper bound for h in the worst case, although h is exponential to k. Usually, the path tree depth h of most of the nodes (i.e., the vertex pairs) is much smaller than the upper bound.

3.4 Putting It All Together

The total process of answering SRJ queries has two phases: offline and online.

Offline phase Given a graph G, we first select some landmarks and store the shortest distances between each vertex and these landmarks. Then, we generate a vertex-pair graph G^p. According to the methods in Section 3.3, we select a h-go cover X over G^p. We store the h-go cover X together with the corresponding SimRank scores in table T_B, as shown in Table 2.

Online phase For any vertex pair $(v_i, v_j) \in U \bowtie V$, we first compute $Sim^u(v_i, v_j)$ (i.e., the upper bound for $Sim(v_i, v_j)$) according to Theorem 3.2. If $Sim^u(v_i, v_j) < \theta$, (v_i, v_j) is pruned safely. Otherwise, we compute $Sim(v_i, v_j)$ by employing Algorithm 1. The vertex pairs whose SimRank scores are no less than θ are returned as answers. The formal description is presented in Algorithm 4.

4. PARTITION-BASED SRJ QUERY

As mentioned above, if graph G is very large, it is not trivial to generate vertex-pair graph G^p due to its large space cost. Li et al. exploit the block structure of graph to compute SimRank scores of all pairwise vertices [18]. The same idea is also employed in computing Pagerank [11]. We propose the partition-based h-go cover index construction and SimRank score computation in this section. Note that, the landmark index has a good scalability up to a graph with billions of vertices [29]. Thus, in this section, we do not discuss the landmark index construction and the upper bound computation in a very large graph, since they are the same with that in Section 3.1.

Algorithm 4 *SRJQuery*(U, V, θ, T_B, G)

Require: two vertex sets U and V in G, threshold θ and the table T_B which stores the SimRank scores of *h-go cover* X_{G^p}.
Ensure: SRJ results
1: $results \leftarrow \emptyset$
2: **for** each vertex pair $(v_i, v_j) \in U \bowtie V$ **do**
3: Compute the $Sim^u(v_i, v_j)$ based on Theorem 3.2
4: **if** $Sim^u(v_i, v_j) \geq \theta$ **then**
5: $Sim(v_i, v_j) \leftarrow QuerySim(v_i, v_j, T_B, G)$ //call Algorithm 1
6: **if** $Sim(v_i, v_j) \geq \theta$ **then**
7: Add (v_i, v_j) into $results$
8: **return** $results$

4.1 Partition-based Approach to Compute SimRank

To make our paper self-contained, this section reviews the partition-based SimRank computation in [18] briefly. Interested readers could refer to [18] for more details.

DEFINITION 4.1. *A graph G is decomposed into n block G_1,...,G_n. A vertex v in block G_i is called a* boundary vertex *if it has one neighbor in another block G_j, where $G_i \neq G_j$. The edge that connects two vertices in two different blocks is called a* crossing edge.

DEFINITION 4.2. *A graph G is decomposed into n blocks G_1,...,G_n. The block-graph G^b has n nodes, where each node corresponds to one block G_i. There is an edge between two nodes in G^b if and only if there are some crossing edges between the corresponding blocks.*

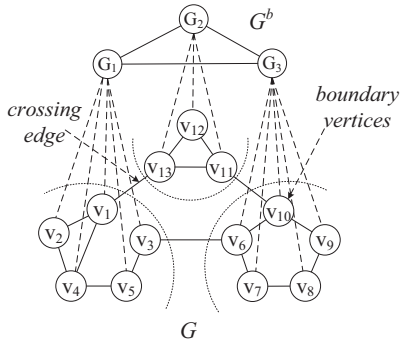


Figure 8: The block structures of a graph

A graph G is decomposed into n blocks G_1,...,G_n. We use MV_{G_i} to denote all boundary vertices in G_i and $Sim_{G^b}(G_i, G_j)$ to denote the SimRank score of two nodes in block-graph G^b, where the two nodes correspond to G_i and G_j, respectively.

Given two vertices v_1 and v_2 of G. If the two vertices are in the same block G_i, we can simply obtain their SimRank score by the procedure QuerySim (Algorithm 1).

$$Sim(v_1, v_2) = Sim_{G_i}(v_1, v_2) \tag{10}$$

where $Sim_{G_i}(v_1, v_2)$ denotes the local SimRank score in block G_i.

Let us consider two vertices v_1 and v_2 in two different blocks G_i and G_j, respectively. We compute $Sim(v_1, v_2)$ as follows:

$$Sim(v_1, v_2) = Sim_{G_i}(v_1, G_i) \cdot Sim_{G^b}(G_i, G_j) \cdot Sim_{G_j}(v_2, G_j) \quad (11)$$

where

$$Sim_{G_i}(v_1, G_i) = \frac{1}{|MV_{G_i}|} \sum_{v_i \in MV_{G_i}} (Sim_{G_i}(v_1, v_i)) \quad (12)$$

$$Sim_{G_j}(v_2, G_j) = \frac{1}{|MV_{G_j}|} \sum_{v_j \in MV_{G_j}} (Sim_{G_j}(v_2, v_j)) \quad (13)$$

Algorithm 5 sketches the query processing of SimRank score of any two vertices. When v_1 and v_2 are in different blocks, we need to compute the SimRank score of v_i and v_j (v_i and v_j are the two corresponding vertices that represent G_i and G_j in the block-graph G^b), and the SimRank scores $Sim_{G_i}(v_1, G_i)$ and $Sim_{G_j}(v_2, G_j)$, and then compute $Sim(v_1, v_2)$ according to Equation (11). Consequently, we can obtain the SimRank score of any two vertices.

Algorithm 5 $QuerySimB(v_1, v_2, T_{B_i}, T_{B_j}, T_{B_b})$

Require: Two vertices v_1 and v_2 in Graph G, $v_1 \in$ block G_i, $v_2 \in$ block G_j, the table T_{B_i} and T_{B_j} with SimRank scores of h-go cover of G_i and G_j respectively, and the table T_{B_b} with SimRank scores of h-go cover of G^b.
Ensure: $Sim(v_1, v_2)$.
1: **if** $G_i = G_j$ **then**
2: $Sim(v_1, v_2) \leftarrow QuerySim(v_1, v_2, T_{B_i}, G_i)$
3: **else**
4: $Sim_{G^b}(G_i, G_j) \leftarrow QuerySim(v_i, v_j, T_{B_b}, G^b)$
5: **for** each $v_x \in MV_{G_i}$ **do**
6: $Sim(v_1, v_x) \leftarrow QuerySim(v_1, v_x, T_{B_i}, G_i)$
7: compute $Sim_{G_i}(v_1, G_i)$ according to Equation (12)
8: **for** each $v_x \in MV_{G_j}$ **do**
9: $Sim(v_2, v_x) \leftarrow QuerySim(v_2, v_x, T_{B_j}, G_j)$
10: compute $Sim_{G_j}(v_2, G_j)$ according to Equation (13)
11: compute $Sim(v_1, v_2)$ according to Equation (11)
12: **return** $Sim(v_1, v_2)$

4.2 Index Building In A Large Graph

According to the computation model in Section 4.1, it is straightforward to extend our method in a large graph. Assume that a graph is partitioned into n blocks G_1,...,G_n, respectively. We generate a block-graph G^b. Then, for each block G_i ($i = 1, ..., n$) and G^b, we employ the methods in Section 3.3 to select h-go covers in each X_{G_i} and X_{G^b}, separately. At running time, given two vertices v_1 and v_2, bases on h-go covers in each block and block-graph, it is easy to compute $Sim(v_1, v_2)$ according to Equations (10) and (11).

5. EXPERIMENTS

In this section, we evaluate our method over large graphs, and compare it with existing solutions, such as LS-join [28], ISP [19] and NI [17]. We also evaluate the effectiveness of SRJ queries in this section.

5.1 Datasets&Setup

We use both synthetic and real datasets. Our method has been implemented using standard C++. The experiments are conducted on a P4 3.0GHz machine with 4G RAM running Linux. The default values of parameters are $\theta = 0.1$, $c = 0.2$, which is the same with the parameter value in [28].

Synthetic Datasets. Two different synthetic graph models are used in our experiments, namely, *E*rdos *R*enyi (ER) and *S*cale *F*ree (SF) models. In ER model, N vertices are connected by M randomly chosen edges. The default values of N and M are 100K and 500K, respectively. The vertex degrees of SF graphs satisfy the power law distribution. We use the graph generator *gengraph win* [4] to generate SF graphs. There are two parameters α and N (the number of vertices) in the generator *gengraph*. The default values of α and N are 2.5 (Usually, $2 < \alpha < 3$ [26]) and 100K, respectively.

Real Datasets. *Yeast* is a protein-protein interaction network in budding yeast, where each vertex represents a protein and an edge denotes the interaction between two proteins. It consists of 2361 vertices and 7182 edges. The proteins are partitioned into 13 clusters according to PIN class information. Each class is assigned a distinct vertex label, namely, there are 13 kinds of labels in the yeast graph.

Cora [22] is a citation network[5], where a vertex represents a paper and an edge denotes the reference relationship between two corresponding vertices. It has 220K vertices and 710K edges. Each vertex (corresponding to one paper) is assigned a label according to the research area it belonging to, such as Database (DB), Artificial Intelligence (AI) and Information Retrieval (IR).

Coauthor is extracted from PROXIMITY DBLP dataset[6]. It is an undirected and unweighted graph, where each vertex corresponds to an author and each edge is introduced if the corresponding authors have at least one co-author paper. Each vertex (author) is assigned a label to denote his research areas, such as, DB, IR and AI. The network has 388K vertices, 1040K edges and 11 vertex labels.

We evaluate the efficiency of our method (Sections 5.2 to 5.4) and compare it with existing solutions, followed by the effectiveness study of SRJ queries in Section 5.5.

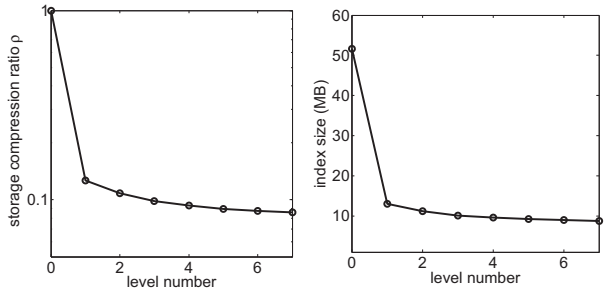
5.2 Evaluating Offline Performance

In our experiment, we find that the set-cover based solution cannot work when $|V(G)| > 1K$, since there are an exponential number of paths to be enumerated. For example, a ER model graph with 1K vertices may have about 500 millions of paths of length 3. The similarity graph-based solution shows better scalability. Here, we introduce a metric storage compression ratio ρ, which is defined as Equation (14) as follows:

$$\rho = \frac{|X|}{|V(G)| \times |V(G)|} \quad (14)$$

where $|X|$ denotes the number of h-go cover nodes. Figure 9(a) shows that ρ decreases with the increasing of the level of similarity graph. Notice that, level 0 means materializing the SimRank scores of all vertex pairs without any storage reduction technique. From Figure 9(a), we know that the number of h-go cover nodes tends to be stable when the level is larger than 3. In Yeast graph, the number of h-go nodes is about $\frac{1}{10}$ of all vertex pairs, as shown in Figure 9(a). Therefore, h-go cover provides an efficient way to reduce the storage cost. In order to handle large graphs, such as Co-author and Cora, we present a partition-based solution. Specifically, we partition a large graph into different blocks by existing partition tool (such as METIS [13]) and each block has about 1K vertices. Table 3 reports the index building time and the index sizes on large graphs, which confirm the good scalability of our method.

[4]http://fabien.viger.free.fr/liafa/generation/
[5]http://people.cs.umass.edu/ mccallum/data.html
[6]http://kdl.cs.umass.edu/data/dblp/dblp-info.html



(a) ρ vs. Similarity Graph Level (b) Index Size vs. Similarity Graph Level

Figure 9: Offline Results on Yeast

Table 3: Index building time and index time on large graphs

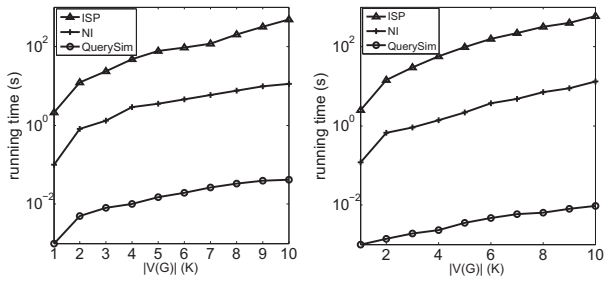
Dataset	$\|V_G\|$	$\|E_G\|$	Index building time(s)	Index size(MB)
*ER*100*K*	100k	400k	343.136	175.597
*SF*100*K*	100k	250k	283.468	127.645
Yeast	2361	7182	80.316	9.579
Cora	220k	710k	1617.356	188.126
Coauthor	338k	1040k	2673.352	948.972

5.3 Evaluating Single-pair SimRank Computation

According to the framework of our SRJ query, we need to compute SimRank scores for all the remaining vertex pairs after pruning. Thus, we design an efficient SimRank computation method for a single vertex pair (Algorithm 1). As mentioned in Section 1, almost all existing methods focus on computing SimRank scores of all vertex pairs except for [17] and [19]. In this section, we compare *QuerySimB* (Algorithm 5) with two existing methods ISP and NI to compute SimRank. Figure 10 shows that our h-go cover-based computation method outperforms ISP and NI by an order of magnitude. Although ISP is designed for computing SimRank of a single-pair, it needs to online enumerate paths between two vertices to compute SimRank score. NI [17] is not as good as our approach, because it is devised to compute a single-source SimRank scores, i.e., the SimRank scores between a given vertex and all other vertices in G.

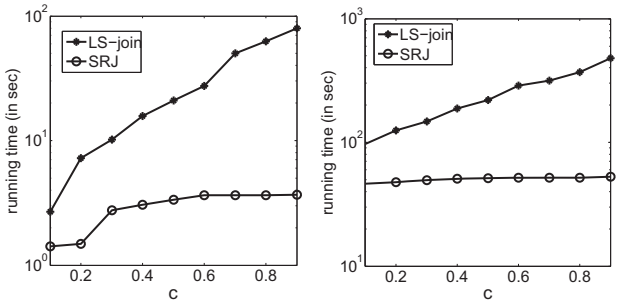
5.4 Evaluating SRJ Query Performance

In this section, we evaluate our method in both synthetic and real datasets and compare it with LS-join [28]. We utilize two different measures, i.e., query response time and pruning power. Given two vertex sets U and V of a graph G, the pruning power τ is defined as



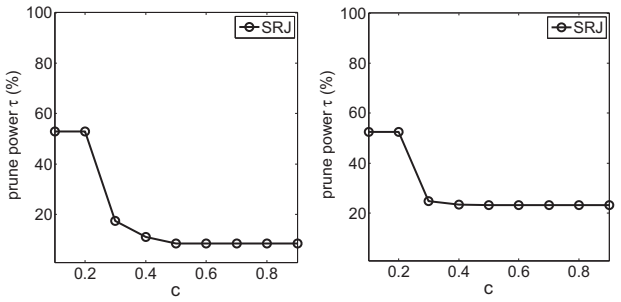
(a) Results on ER graphs (b) Results on SF graphs

Figure 10: Running time vs. $|V_G|$



(a) Results on Yeast (b) Results on Coauthor

Figure 11: Running time vs. c



(a) Results on Yeast (b) Results on Coauthor

Figure 12: Prune power τ vs. c

follows:

$$\tau = \frac{filter_{number}}{|U| \times |V|} \tag{15}$$

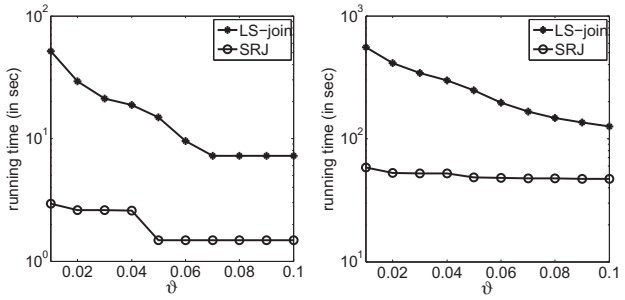
where $filter_{number}$ denotes the number of vertex pairs that can be pruned by Theorem 3.2 without computing their SimRank scores. $|U|$ and $|V|$ denotes the number of vertices (in G) in U and V, respectively.

As LS-join works on the top-k queries and our method works on the threshold queries, the two methods cannot be compared directly. To enable comparison, we first perform our method to find some vertex pairs whose SimRank scores are not less than the threshold θ. Then, we set the parameter k in LS-join method as the number of delivered matches by our method.

5.4.1 Effect of the Decay Constant c

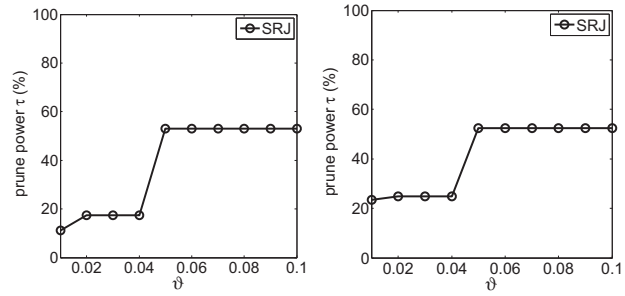
To study the effect of c, we fix the threshold θ to be 0.1 and vary c from 0.1 to 0.9. As shown in Figure 11, the gap of the running time between SRJ and LS-join increases rapidly with increasing of the decay factor c. Though the decay factor may affect the estimated upper bound, which results in more candidate vertex pairs, the SimRank computation process of our method is independent of the decay factor c. However, LS-join is based on the iterative computation and its convergent rate is determined by c.

τ describes the prune power. As shown in Figure 12, when the decay constant c is small, more than half of all the the vertex pairs can be pruned in Yeast and Coauthor. As c increases, the prune power decreases, because the SimRank scores between the vertex pairs increase with respect to c. When c is larger than 0.5, the prune power changes a little, because a pair of vertex can be pruned only if their shortest path distance is longer than 12 according to Theorem 3.2. Thus, only a small proportional of the vertex pairs can satisfy this constraint, which can be pruned.



(a) Results on Yeast (b) Results on Coauthor

Figure 13: Running time vs. θ



(a) Results on Yeast (b) Results on Coauthor

Figure 14: Prune power τ vs. θ

5.4.2 *Effect of the Threshold θ*

To study the effect of threshold θ, we fix the decay constant to be 0.1 and vary the θ from 0.01 to 0.1. Figure 13 depicts the running time with regard to θ. The running time of both our method (denoted as SRJ) and LS-join decrease as θ increases. One of the reasons is that the number of join results decreases. Figure 13 shows that SRJ outperforms LS-join by almost an order of magnitude.

As shown in Figure 14, the number of vertices of pruned by the estimated upper bound increases as the threshold increases. When threshold is set to be 0.05, all vertex pairs with estimated shortest path distance no less than 3 are pruned according to Theorem 3.2. Even though the threshold increases from 0.05 to 0.1, we still cannot prune the vertex pairs whose estimated shortest path distances are less than 3. Thus, the pruning power is stable when the threshold increases from 0.05 to 0.1, as shown in Figure 14.

5.5 Effectiveness Evaluation

To prove the effectiveness of SRJ, we analyze the results over two real datasets Cora and Coauthor. In this section, we focus on the case studies of SRJ queries to check whether the results returned by SRJ are reasonable. Evaluating the superiorities of SimRank is beyond this article's scope, and it has been fully studied in [10].

Finding similar papers in cross-disciplinary studies in Cora networks. The aim is to find pairs of paper: (1) The two papers are derived from two areas (such as IR and DB), and (2) they focus on similar problems. Figures 15(a) and 15(b) shows two examples. In each example, vertex sets U and V have three vertices (in Cora network) corresponding to three papers, respectively. Note that the class labels of papers are assigned by authors of Cora data [22]. Let us see an example in Figure 15(a). One pair is "Knowledge integration for structured information sources containing text" (from IR area) and "Heuristic joins to integrate structured heterogeneous data" (from DB area). They both focus on the data integration problem. More examples can be found in Figure 15(a) and (b).

Finding researchers that share the similar research interests in DBLP networks. Given two groups of authors, we want to find some pairs of authors who share the similar research interests. A case study is presented in Figure 15(c). One returned pair is Cai-Nicolas Ziegler and Karen H. L. Tso-Sutter. Cai-Nicolas Ziegler is interested in social network and recommendation, while Karen H. L. Tso-Sutter focuses on the recommender systems. Note that the two authors do not have collaboration in DBLP data. We report five author pairs in Figure 15(c).

6. RELATED WORK

Graph model has attracted extensive attentions in the database community. Many research problems over graphs have been proposed, such as, subgraph search [32, 34], approximate subgraph search [33, 15], shortest-path query [3], graph pattern match [4, 35] and similarity join query [28]. In this paper, we focus on the problem of SimRank-based similarity join.

SimRank is a link-based similarity measure [10], which is applicable to any domain with object-to-object relationships. Many algorithms have been proposed to improve the efficiency [21, 8, 17, 18, 19]. Generally speaking, they can be divided into three categories, as discussed in Section 1.

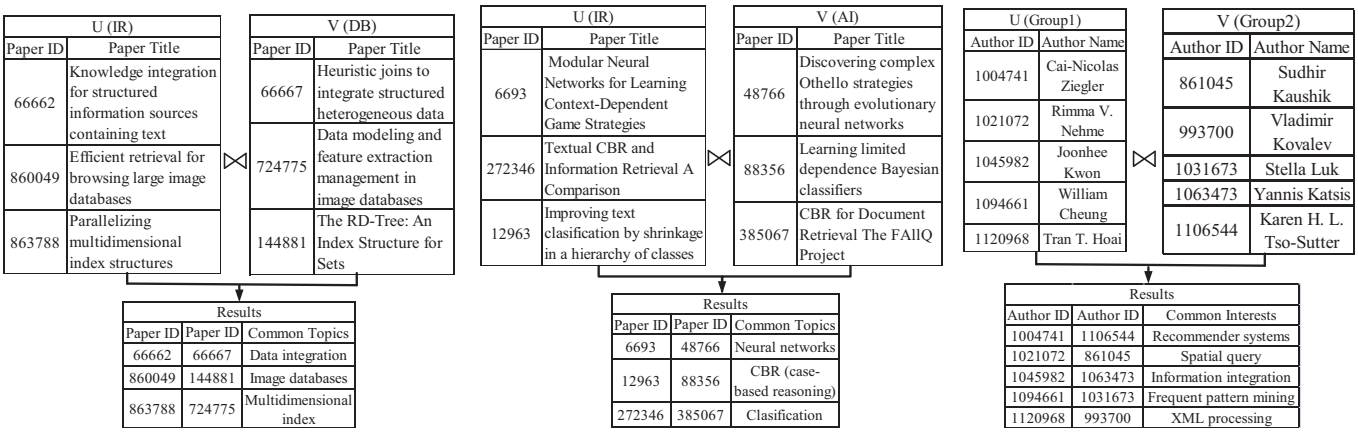
The first one is the methods that compute all-pairwise SimRank. Most existing methods focus on computing SimRank scores between all-pairwise vertices. D. Lizokin et al. propose some optimization techniques to improve the time complexity from $O(N^4)$ to $\min(O(N \cdot M), O(\frac{N^3}{\log_2 N}))$, where M denotes the number of edges in graph G [21]. It also introduces a threshold sieving heuristic. To handle large graphs, the authors of [8] compute approximate SimRank scores by using a database of fingerprint trees, which is a compact representation of precomputed random walks. Li et al. propose a partition-based method BlockSimRank to compute SimRank scores with time complexity $O(N^{4/3})$ [18]. It partitions the graph into blocks and computes the local and global similarities respectively.

The second category is the method computing single-source SimRank. Li et al. rewrite the SimRank equation into a non-iterative form, based on which a family of approximate SimRank computation algorithms are developed. However, the space cost of this method is $O(N^2)$ [17]. The third category is the method computing single-pair SimRank. The authors of [19] propose to compute SimRank of a single-pair vertices by online enumerating all paths rooted at these two vertices, which is quite expensive especially for large graphs.

The work in [28] is the most closely related one to our work in this paper. They propose a link-based similarity join with respect to an *e*-function generalizing PageRank and SimRank. However, it adopts the iterative computation model to online compute similarities. We make extensive experiments on both synthetic and real datasets, which confirm that our method outperforms the method in [28] significantly.

7. CONCLUSIONS

In this paper, we focus on the SimRank-based join query problem over large graphs. To reduce the search space, we first design a novel upper bound for the SimRank score between two vertices in a graph. In order to compute the SimRank score between two vertices, we propose a "h-go cover" index and present a non-iterative computation model to compute the SimRank. Specifically, we only need to materialize a small proportion of the SimRank scores, based



(a) IR and DB (b) IR and AI (c) Case Study over Coauthor Dataset

Figure 15: Effectiveness Case Study

on which we can recover all the other SimRank scores which are not materialized. We prove that finding minimum h-go cover is NP-hard and propose two heuristic methods to select h-go cover. We evaluate our methods on both synthetic and real datasets. Extensive experiments show that our method outperforms existing methods greatly.

8. REFERENCES

[1] Z. Abbassi and V. S. Mirrokni. A recommender system based on local random walks and spectral methods. In *WebKDD/SNA-KDD*, pages 139–153, 2007.

[2] Aristotle. Rhetoric. nichomachean ethics. *Rackman transl. Cambridge: Harvard Univ. Press*, 23, 1934.

[3] E. P. F. Chan and H. Lim. Optimization and evaluation of shortest path queries. *VLDB J.*, 16(3):343–369, 2007.

[4] J. Cheng, J. X. Yu, B. Ding, P. S. Yu, and H. Wang. Fast graph pattern matching. In *ICDE*, pages 913–922, 2008.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press.

[6] I. Dinur and S. Safra. On the hardness of approximating minimum vertex cover. *Annals of Mathematics*, 162(1):439–485, 2005.

[7] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. Adding regular expressions to graph reachability and pattern queries. In *ICDE*, pages 313–338, 2011.

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

[9] J. E. Gentle. Gaussian elimination. In *Numerical Linear Algebra for Applications in Statistics*, pages 87–91, 1998.

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

[11] S. D. Kamvar, T. H. Haveliwala, C. D. Manning, and G. H. Golub. Exploiting the block structure of theweb for computing pagerank. In *Technical Report,Stanford*, 2003.

[12] G. Karakostas. A better approximation ratio for the vertex cover problem. *ACM Transactions on Algorithms*, 5(4), 2009.

[13] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *J. Parallel Distrib. Comput.*, 48(1), 1998.

[14] M. M. Kessler. Bibliographic coupling extended in time: Ten case histories. *Information Storage and Retrieval*, 1(4):169–187, 1963.

[15] A. Khan, N. Li, X. Yan, Z. Guan, S. Chakraborty, and S. Tao. Neighborhood based fast graph search in large networks. In *SIGMOD Conference*, pages 901–912, 2011.

[16] S. Khot and O. Regev. Vertex cover might be hard to approximate to within 2-epsilon. *J. Comput. Syst. Sci.*, 74(3):335–349, 2008.

[17] 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.

[18] P. Li, Y. Cai, H. Liu, J. He, and X. Du. Exploiting the block structure of link graph for efficient similarity computation. In *PAKDD*, pages 389–400, 2009.

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

[20] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American Society for Information Science and Technology*, 58:1019–1031, 2007.

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

[22] A. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval Journal*, 3:127–163, 2000.

[23] M. McPherson, L. S. Lovin, and J. M. Cook. Birds of a feather: Homophily in social networks. *Annual Review of Sociology*, 27:415–444, 2001.

[24] Plato. Laws. plato in twelve volumes. *Bury translator. Cambridge: Harvard Univ. Press*, 11, 1968.

[25] M. Potamias, F. Bonchi, C. Castillo, and A. Gionis. Fast shortest path distance estimation in large networks. In *CIKM*, pages 867–876, 2009.

[26] Réka Albert and Albert-László Barabási. Statistical mechanics of complex networks. *Reviews of Mordern Physics*, 74:47–97, 2002.

[27] H. Small. Co-citation in the scientific literature: A new measure of the relationship between two documents. *Journal of the American Society for Information Science*, 24:265–269, 1973.

[28] L. Sun, R. Cheng, X. Li, D. W. Cheung, and J. Han. On link-based similarity join. *PVLDB*, 4(11):714–725, 2011.

[29] K. Tretyakov, A. Armas-Cervantes, L. García-Bañuelos, J. Vilo, and M. Dumas. Fast fully dynamic landmark-based estimation of shortest path distances in very large graphs. In *CIKM*, pages 1785–1794, 2011.

[30] S. Trißl and U. Leser. Fast and practical indexing and querying of very large graphs. In *SIGMOD*, pages 845–856, 2007.

[31] H. Wang, H. He, J. Yang, P. S. Yu, and J. X. Yu. Dual labeling: Answering graph reachability queries in constant time. In *ICDE*, pages 845–856, 2006.

[32] S. Zhang, S. Li, and J. Yang. Gaddi: distance index based subgraph matching in biological networks. In *EDBT*, pages 192–203, 2009.

[33] S. Zhang, J. Yang, and W. Jin. Sapper: Subgraph indexing and approximate matching in large graphs. *PVLDB*, 3(1):1185–1194, 2010.

[34] P. Zhao and J. Han. On graph query optimization in large networks. *PVLDB*, 3(1):340–351, 2010.

[35] L. Zou, L. Chen, and M. T. Özsu. Distancejoin: Pattern match query in a large graph database. *PVLDB*, 2(1):886–897, 2009.