Hop-constrained Subgraph Query and Summarization on Large Graphs

Yu Liu, Qian Ge, Yue Pang, and Lei Zou

Peking University, Beijing, China
{dokiliu, geqian, michelle.py, zoulei}@pku.edu.cn

Abstract. We study the problem of hop-constrained relation discovery in a graph, i.e., finding the structural relation between a source node s and a target node t within k hops. Previously studied $s-t$ graph problems, such as distance query and path enumeration, fail to reveal the $s-t$ relation as a big picture. In this paper, we propose the k-hop $s-t$ subgraph query, which returns the subgraph containing all paths from s to t within k hops. Since the subgraph may be too large to be well understood by the users, we further present a graph summarization method to uncover the key structure of the subgraph. Experiments show the efficiency of our algorithms against the existing path enumeration based method, and the effectiveness of the summarization.

Keywords: Hop-constrained subgraph query · k-hop $s-t$ subgraph · $s-t$ graph summarization.

1 Introduction

With the advent of graph data, it has become increasingly important to manage large-scale graphs in database systems efficiently. Generally, vertices in graphs represent entities and edges the relations between them. Paths, formed by chaining together multiple edges that share vertices, can be seen as representing more complex relations between its source and destination vertices. The fundamental problem of discovering the relation between two entities has thus given rise to numerous path-finding algorithms, the majority of which aims at determining whether a relation exists between two vertices (i.e., reachability) or finding a relation between two vertices that satisfy specific properties (e.g., shortest path and top-k path enumeration). However, in certain applications, focusing on one relation (path) at a time is not enough. We list two real-world scenarios in which the $s-t$ relation is demanded as a big picture.

Motivation 1. Discovery of ownership structure. In an equity network, vertices represent corporations, an edge points from a corporation to another if the former holds shares of the latter. An important query would be to discover the ownership structure between two corporations, characterized by chains of shareholding that may span across the whole network. The results of such queries can help gain insights into a market's dynamics, e.g., how financial risks propagate, and therefore help with risk management.

Motivation 2. Relation discovery in social networks. In a social network, vertices represent persons and edges their relationships, which may include *follower-of*, *friend-of*, *parent-of*, etc. A query may aim to obtain the "social group" formed with two persons of interest as the source and destination respectively, composed of other persons that act as intermediates for the former to reach the latter. Results of such queries may benefit social network analysis (e.g., for advertising) and anomaly detection (e.g., for detecting crimes and terrorism).

Unfortunately, these applications cannot be appropriately handled by existing path-finding problems and their solutions, for they call not for single paths, but a *subgraph* that merges all relevant relations between the source and the destination. In this paper, we tackle the problem of efficiently computing a subgraph that represent the relations between a source and a destination vertex. Intuitively, given a hop constraint k, we compute a subgraph containing the paths from s to t within k hops, which is referred to as the k-hop $s-t$ subgraph. Specifically, several algorithms based on graph traversal and pruning techniques are developed to compute the subgraph. Considering the subgraph may be too large to be well understood by users (e.g., for visualization) for large graphs, we further propose a graph summarization technique to only reveal the structural skeleton of the subgraph. The main contribution of our paper is summarized as follows.

- We first propose the *k-hop $s-t$ graph query*, which returns a subgraph containing all paths from s to t within k hops. Compared to existing queries such as (k-hop) reachability, $s-t$ distance query and path enumeration, the subgraph query reveals the $s-t$ relation as a big picture. We also propose a traversal-based algorithm which is worst-case optimal in answering subgraph queries.
- Based on the result subgraph, we further propose the notion of *$s-t$ graph summarization with hop constraint*, which contracts the subgraph into a summarized graph with only a few nodes (controlled by a user-defined parameter). We present an algorithm based on skeleton node selection and local graph clustering, and demonstrate two skeleton node selection strategies which depend on path frequency and walking probability, respectively.
- On several large graph datasets we demonstrate the efficiency of our subgraph finding algorithm against the baselines based on path enumeration, as well as the effectiveness of our algorithms in terms of $s-t$ relation discovery and subgraph summarization.

The remainder of the paper is organized as follows. Section 2 gives the formal definitions of our studied problems. We discuss related work in Section 3, including several baseline methods. In Section 4 and 5, we propose our solutions for the k-hop subgraph query and hop-constrained $s-t$ graph summarization, respectively. Section 6 reports the experimental results. We conclude the paper in Section 7.

2 Preliminaries

2.1 Problem Statement

We first give several formal definitions about paths and subgraphs. Then we describe the two studied problems.

Definition 1 (Path and k-hop path). *Given a simple and directed graph $G=(V,E)$, a path $p=(v_1=s,v_2,\ldots,v_l=t)$ in G is defined as a sequence of edges, i.e.,*

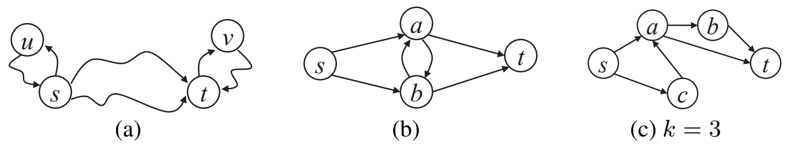


Fig. 1. An illustration of the definition of k-hop $s - t$ subgraph.

$(v_i, v_{i+1}) \in G.E, \forall i \in [1, l)$. *Note that the length of* p *is* $l - 1$, *and* p *is referred to as a* $(l - 1)$*-hop path.*

We say path p contains a cycle if there exists some $1 \leq i < j \leq l$ such that $v_i = v_j$.

Definition 2 (Union of paths). *Given a set of paths* $\{p_1, \ldots, p_m\}$, *where each path* $p_i = (v_{1_i} = s, \ldots, v_{l_i} = t)$ *is from* s *to* t. *Subgraph* $G_{st} = (V_{st}, E_{st})$ *is a union of paths* $\{p_1, \ldots, p_m\}$, *if* $V_{st} = \cup_{i \in [1,m]} \{v_{1_i} \cup \ldots \cup v_{l_i}\}$, *and* $E_{st} = \cup_{i \in [1,m]} \{(v_{j_i}, v_{(j+1)_i}), j \in [1, l)\}$. *Duplicated edges are removed during the union operation.*

Definition 3 (k-hop $s - t$ subgraph). *A subgraph* G_{st} $(s \neq t)$ *is referred to as* k*-hop* $s - t$ *subgraph if it is the union of* all k*-hop* $s - t$ *paths, such that for each path* $p = (v_1 = s, \ldots, v_l = t)$, *(1)* $v_i \neq s, \forall i \in (1, l]$; *and (2)* $v_j \neq t, \forall j \in [1, l)$. *We also refer to* G_{st} *as* k-hop subgraph, *or* subgraph *when the context is clear.*

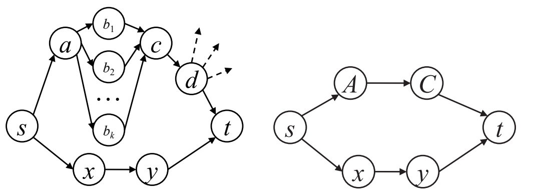
The definition of subgraph query aims to reveal the k-hop relation between s and t as a whole, rather than enumerating separate paths. However, we are not interested in (1) nodes only reachable to t via s (u in Fig. 1(a)), or (2) nodes only reachable from s via t (v in Fig. 1(a)). Intuitively, u and v do not contribute to the relation between s and t, therefore we do not take them into consideration.

Nonetheless, we allow certain cycles in the $s - t$ relation. For example, after inserting two 3-hop paths (s, a, b, t) and (s, b, a, t) into the subgraph (Fig. 1(b)), a cycle is formed between a and b. Such cycle may represent meaningful relations, for instance, the circulating ownership of stock in financial networks. Also note that G_{st} may contain $s - t$ path longer than k ((s, c, a, b, t) in Fig. 1(c)), which is inevitable because of the union of different paths. We ignore these longer paths in that G_{st} only focuses on the close (i.e., k-hop) relation between s and t.

In this paper, we study the problems of k-hop $s - t$ subgraph query and k-hop subgraph summarization, defined as follows.

Definition 4 (Hop-constrained subgraph query). *Given a directed graph* $G = (V, E)$, *a source node* s, *a target node* $t (t \neq s)$, *and the hop constraint* k, *return the* k*-hop* $s - t$ *subgraph* G_{st}^k.

For simplicity, we only consider simple directed graphs in this paper. Moreover, the relation defined above points from s to t and is asymmetric (following out-edges). Note that the problem setting can be easily extended to other types of graphs (e.g., undirected or weighted graphs), while the relation can be defined based on a set of paths following in-edges, or allowing a mixture of outgoing and incoming edges.



(a) Subgraph G_{st} (b) The skeleton graph G_{st}^{sum}

Fig. 2. k-hop subgraph and its summarization.

Remarks. As mentioned by previous work [21, 23], posing a constraint on the number of hops is reasonable, in that the strength of the relation drops dramatically with distance. Nonetheless, our studied problem is still well-defined when k is set to ∞. Let G_{SCC} be the directed acyclic graph (DAG) where each node in G_{SCC} corresponds to a strongly connected component of G. Let C_s (resp. C_t) be the strongly connected component containing s (resp. t). To this end, we are essentially extracting the subgraph in G which corresponds to a subgraph in G_{SCC} composed by all paths from C_s to C_t.

Although the k-hop $s-t$ subgraph provides a way to understand the relation between s and t, the size of the subgraph can be extremely large, e.g., with hundreds of thousands of vertices for a reasonable k (say 6) and a medium-sized graph. This prevents us from finding the underlying structure of $s-t$ relation. Therefore, we also consider the problem of subgraph summarization, which summarizes the result subgraph into a small and succinct one.

We adopt a skeleton node-based summarization method that contains two steps. First, we find a set of skeleton nodes that play important role in the underlying structure, where the number of skeleton nodes is a user-defined parameter. Then, we conduct local graph clustering from these skeleton nodes. Other nodes in the subgraph (except s and t) is assigned to one of the communities C_i corresponding to some skeleton node v_i. In particular, given a k-hop s-t subgraph G_{st} and a set of skeleton nodes V_S (detailed later), the skeleton graph $G_{st}^{k,h}$ is the summarization graph of G_{st}^{k}, where $G_{st}^{k,h}.V = V_S \cup \{s,t\}$, and for any $u, v \in G_{st}^{k,h}.V$, $(u,v) \in G_{st}^{k,h}.E$ if some criterion is satisfied, e.g., there exists an edge $(x,y) \in G_{st}^{k}.E$ s.t. $x \in C_u$ and $y \in C_v$, or the probability of node u reaching node v in G_{st}^{k} is above some threshold.

Definition 5 (Hop-constrained $s-t$ graph summarization). *Given a directed graph $G=(V,E)$, a source node s, a target node t, the hop constraint k, and the number of skeleton nodes h, return the k-hop $s-t$ summarized graph $G_{st}^{k,h}$, which contains h super-nodes corresponding to h local communities in the $s-t$ subgraph G_{st}^{k}.*

Fig.2 demonstrates a skeleton graph of the k-hop subgraph. We set the number of skeleton nodes as 4. Intuitively, node a, c, and d are more important than $b_1, \dots, b_k$ (e.g., shell companies in financial networks) because more paths go through them. Node a and c are preferred over d because the latter is a hot point (i.e., node of large degree, shown in dashed edges), but contributes few edges to G_{st}. Therefore, we compress node a and $b_1, \dots, b_k$ to a super-node A, c and d to a super-node C. The summarization graph highlights the structural information in the $s - t$ relation and is easier to visualize. We will discuss the strategies for finding skeleton nodes in Section 5.2.

3 Existing Work

To the best of our knowledge, there is no existing work that directly considers the problem of $s - t$ subgraph query or summarization. However, a bunch of works study similar problems that are more or less aimed at determining the relation between a pair of vertices, in which the techniques used can be extended to our problem settings. We categorize them as follows and discuss their relation to our problems.

3.1 Reachability / k-hop Reachability

The classic reachability problem studies whether there exists a path from a source vertex to a destination vertex. The majority of existing reachability algorithms [24, 25, 28, 30] are index-based and focus on the directed acyclic graph (DAG) contraction of the input graph. Some generalized versions of reachability queries have also been proposed, including the label constrained reachability [16] and the k-hop reachability [7], which answers reachability with the hop constraint k. They can not be directly applied to our problem because too few information between s and t is preserved.

3.2 Shortest Path / k-shortest Paths

A plethora of works study the single-pair shortest path problem [3–5, 8, 11, 12, 14], as well as the k-shortest paths (KSP) problem [6, 10, 15, 18, 29] which returns the top-k shortest paths between the source and the target. Though the algorithms for KSP can be used for path enumeration, it has been proved inefficient [13].

3.3 Path Enumeration / Top-k path enumeration

The path enumeration problem aims to find all paths from the source to the target, possibly with additional constraints (e.g., hop constraint). To answer the subgraph query, we can first enumerate all (k-hop) paths and then combine them into a subgraph. We first show that depth-first search (DFS) can be easily applied to answer k-hop subgraph queries.

The pseudo-code is illustrated in Alg. 1. Given a directed graph G, a source node s, a target node t, and the hop constraint k, the algorithm first initialize G_{st}^{k} as empty graph (Line 1) and then invoke k-DFS (Line 2). The procedure k-DFS traverses the graph from s in a depth-first way. Once it reaches t, which means a path from s to t is found, we insert the path into G_{st}^{k} (Lines 6-7); then we stop traversal immediately, ignore any node only reachable from s via t (Line 8). Note that the traversal is limited within k-hops from s (Lines 10-12). We have the following theorem.

Theorem 1. *Algorithm 1 correctly computes the k-hop subgraph with $O(n^k)$ worst time complexity.*

Algorithm 1 The Baseline Algorithm

```
Input: Directed graph $G = (V, E)$; Source $s$; Target $t$; Hop constraint $k$
Output: $G^k_{st}$, the $k$-hop $s - t$ subgraph
1: Initialize $G^k_{st}$ as empty graph and stack $\mathcal{S} = \emptyset$;
2: $k$-DFS($G, s, s, t, k, \mathcal{S}, G^k_{st}$);
3: return $G^k_{st}$;
4: Procedure $k$-DFS(Graph $G$, Current node $u$, Source $s$, Target $t$, Hop constraint $k$, Stack $\mathcal{S}$,
   Partial subgraph $G^k_{st}$)
5:     Push $u$ to $\mathcal{S}$;
6:     if $u = t$ then
7:         Add $p(\mathcal{S})$ to $G^k_{st}$;
8:         return ;
9:     if $k > 0$ then
10:        for each $v \in O(u)$ and $v \neq s$ do
11:            $k$-DFS($G, v, s, t, k - 1, \mathcal{S}, G^k_{st}$);
```

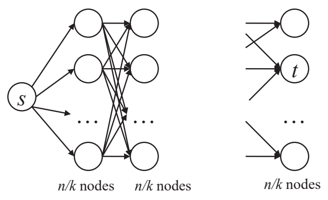


Fig. 3. Worst case graph for k-DFS.

Proof. The correctness of Alg. 1 can be easily derived by the property of DFS and the definition of k-hop subgraph. To show the algorithm runs in $O(n^k)$ in worst case, consider the following graph (Fig. 3). Since there are totally $\left(\frac{n}{k}\right)^k$ $k - hop$ paths from s, the time complexity of DFS is $O(k \cdot \left(\frac{n}{k}\right)^k) \sim O(n^k)$.

To improve the practical efficiency of k-DFS, in the theoretical paper [13], they propose T-DFS, a polynomial delay algorithm which takes $O(km)$ to find one path. Recent work [23] aims at detecting all simple cycles within k hops after the insertion of an edge on dynamic graphs, by enumerating all simple paths within $k - 1$ hops between the two vertices that the new edge is adjacent to. To speed up query processing, it employs a hot point index to prevent repetitive traversals from vertices with high degrees. On the other hand, [21] employs a pruning-based algorithm to speed up k-hop simple path enumeration. Note that any path enumeration algorithm has at least $\Omega(k\delta)$ complexity, where δ is the number of valid paths. Since there can be tremendous numbers of paths, they are not suitable to answer top-k subgraph queries of which the answer size is bounded by $O(n + m)$. Besides, current methods do not consider cycles for simplicity, whereas we include some types of cycles that also represent the relationship in the subgraph.

Other related works, such as graph summarization [9, 17, 22, 26, 27], primarily consider summarizing the whole input graph or the subgraph around a given node s while preserving some properties, and are not specially tailored for the $s-t$ relation discovery.

4 KHSQ: The K-hop Subgraph Query Algorithm

4.1 Rationale

Recall that the baseline algorithm incurs $O(n^k)$ cost for k-hop subgraph query because each node v may conduct DFS multiple times. In this section, we propose a simple yet effective algorithm that accelerates the querying speed by utilizing the distance information. Intuitively, we first conduct a k-hop breadth first search (BFS) to compute $d^f(s,v)$ for each v, which is the distance from s to v. Similarly, we compute the distance of v to t (denoted by $d^b(v,t)$) by a traversal from t following in-edges. Then, the distance information are used to prune most repeated or unnecessary traversals.

4.2 Algorithm

Our algorithm, denoted as KHSQ ($\underline{\text{K}}$-$\underline{\text{H}}$op $\underline{\text{S}}$ubgraph $\underline{\text{Q}}$uery), is demonstrated in Alg. 2. We first invoke k-BFS both from s and t to compute the distance array (Lines 2-3). The k-BFS procedure (Lines 7-18) is analogous to the breadth-first search, but only traverses k levels. Then we invoke DFS-SQ (Line 5), which improves the naive k-DFS in two aspects:

First, instead of enumerating all k-hop paths and union them together, we check for each edge (u,v) to see if it is in G_{st}. To be precise, recall that in the definition of k-hop subgraph G_{st}, $(u,v) \in E_{st}$ if there exists some k-hop path containing it. If $d^f(s,u)+1+d^b(v,t) \leq k$, we are sure path $p = (p^*(s,u),(u,v),p^*(v,t))$ is a k-hop path, where $p^*(s,u)$ (resp. $p^*(v,t)$) denotes one shortest path from s to u (resp. from v to t). Hence, it is sufficient to check each edge only once.

Second, we also guarantee that each node (and its out-edges) is visited only once. Each node v conducts neighbor traversal only if the path in stack $\mathcal{S}$ is a shortest path to v, and v has not conducted the traversal before. Since our algorithm checks each node and each edge at most once, the complexity is asymptotically linear of the problem inputs.

4.3 Analysis

The following theorem states the correctness of KHSQ.

Theorem 2. *Algorithm 2 correctly finds the k-hop subgraph.*

Proof. It is easy to see that procedure k-BFS correctly computes the distance from s to v within k-hops, and set the distance of other nodes as ∞. This also holds for the traversal from t on G^r, which is in fact traversing in-edges of G. To show the correctness of procedure DFS-SQ, note that every edge (u,v) in every k-hop path will be added to G_{st} according to our checking condition (Line 27 of Alg. 2). On the other hand, if some edge (u',v') is not contained by any such path, then we have $d^f(s,u)+1+d^b(v,t) > k$ and it will be excluded. Path like $v \to s \to t$ (or $s \to t \to v$) and with less than k-hops

Algorithm 2 KHSQ

```
Input: Directed graph $G = (V, E)$; Source $s$; Target $t$; Hop constraint $k$
Output: $G_{st}$, the $k$-hop $s - t$ subgraph
 1: Initialize $G_{st}$ as empty graph and stack $\mathcal{S} = \emptyset$;
 2: $d^f(s, *)$ = $k$-BFS($G, s, t, k$);
 3: $d^b(*, t)$ = $k$-BFS($G^r, t, s, k$);
 4: Initialize $dfs(v) = false, \forall v \in V$;
 5: DFS-SQ($G, s, s, t, k, \mathcal{S}, d^f(s, *), d^b(*, t), dfs, G_{st}$);
 6: return $G_{st}$;
 7: Procedure $k$-BFS(Graph $G$, Source $s$, Target $t$, Hop constraint $k$)
 8:     Initialize $d(s, s) = 0$, $d(s, v) = \infty$ for $\forall v \in V \backslash \{s\}$;
 9:     Initialize queue $\mathcal{Q} = \{s\}$;
10:     Initialize $lvl = 0, numThisLvl = 1, numNextLvl = 0$, $visited(s) = true$, and
        $visited(v) = false, \forall v \in V \backslash \{s\}$;
11:     while $\mathcal{Q} \neq \emptyset$ do
12:         for $i = 1$ to $numThisLvl$ do
13:             $u$ = remove($\mathcal{Q}$);
14:             if $u \neq t$ and $lvl < k$ then
15:                 for each $v \in O(u)$ and $v \neq s$ do
16:                     if $visited(v) = false$ then
17:                         Add $v$ to $\mathcal{Q}$, $d(s, v) = d(s, u) + 1, visited(v) =$
                            $true, numNextLvl + +$;
18:         $lvl + +, numThisLvl = numNextLvl, numNextLvl = 0$;
19: Procedure DFS-SQ(Graph $G$, Current node $u$, Source $s$, Target $t$, Hop constraint $k$, Stack
    $\mathcal{S}$, forward distance $d^f(s, *)$ and backward distance $d^b(*, t)$, Label $dfs$, Partial subgraph
    $G_{st}$)
20:     Push $u$ to $\mathcal{S}$;
21:     $dfs(u) = true$;
22:     if $u = t$ then
23:         pop($\mathcal{S}$);
24:         return ;
25:     for each $v \in O(u)$ and $v \neq s$ do
26:         if $d^f(s, u) + 1 + d^b(v, t) \leq k$ then
27:             Add edge $(u, v)$ to $G_{st}$;
28:         if $|S| = d^f(s, v)$ and $dfs(v) = false$ then
29:             DFS-SQ($G, v, s, t, k, \mathcal{S}, d^f(s, *), d^b(*, t), dfs, G_{st}$);
30:     pop($\mathcal{S}$);
```

is eliminated by setting $d^b(v, t)$ (or $d^f(s, v)$) as ∞ in k-BFS. Therefore, the subgraph returned by KHSQ algorithm is equal to the union of all valid k-hop paths, and the correctness follows.

We bound the time and space complexity of KHSQ as follows. Since procedure k-BFS only conducts breath-first search from s and within k hops, its time complexity is $O(n + m)$. As discussed above, we have demonstrated that procedure DFS-SQ visits each node and each edge at most once. Therefore, the cost is still bounded by $O(n +$

m). The following theorem states that KHSQ is highly efficient in answering k-hop subgraph queries.

Theorem 3. *The time complexity of KHSQ is $O(n+m)$, which is asymptotically linear with the problem inputs and worst-case optimal.*

Proof. Since KHSQ only invokes k-BFS twice and DFS-SQ from s once, the time complexity can be easily derived. To see the algorithm is worst-case optimal, consider the case that $|G_{st}| = \Theta(|G|)$, where $|G| = |G.V| + |G.E| = n + m$, e.g., $G_{st} = G$. Since each node and edge must be processed with $O(1)$ cost, our claim holds.

5 KHGS: The k-hop $s - t$ Graph Summarization Algorithm

5.1 Problem Overview

Though the k-hop subgraph demonstrates the relation between s and t as a whole, it suffers from extremely large size for many real-world networks. For example, on a medium-sized graph, say, with millions of nodes and a reasonable *k* (e.g., 6), the result subgraph may contain hundreds of thousand nodes, which prevents us from understanding the underlying structure of the $s - t$ relation. Hence, we propose a skeleton node based method for subgraph summarization, which relies on a set of *skeleton nodes* (plus s and t) while the edges and paths between them are contracted and summarized (Recall its definition in Sec. 2.1.). As long as we correctly select the most important nodes as the skeleton of G_{st}, the summarized graph reveals the key structure of $s - t$ relation hidden in a bunch of edges.

Our algorithm framework is shown in Alg. 3. Given a graph G, a source node s, a target node t, and the hop constraint k, the KHGS (K-Hop Graph Summarization) algorithm first invokes KHSQ to compute the k-hop subgraph G^k_{st}. To get the summarization graph, our algorithm contains two steps. We first choose h most important nodes (referred to as skeleton nodes) from G^k_{st}, where h is a user-defined parameter. Then we contract G^k_{st} into $G^{k,h}_{st}$, which can be simply implemented by h distinct local traversals (e.g., clustering) from the skeleton nodes. Specifically, we present two importance measures based on path frequency and walking probability, respectively. We describe the algorithms for skeleton node selection in the following subsection.

Algorithm 3 KHGS

```
Input: Graph $G$; Source $s$; Target $t$; Hop constraint $k$; Number of skeleton nodes $h$
Output: $G^{k,h}_{st}$, the skeleton graph of the $s - t$ subgraph $G_{st}$
1: $G^k_{st}$ = KHSQ($G, s, t, k$);
2: $V_S$ = FindSkeletonNodes($G^k_{st}, s, t, k, h$);
3: $G^{k,h}_{st}$ = SummarizedGraphConstruction($G^k_{st}, s, t, k, V_S$);
4: return $G^{k,h}_{st}$;
```

Algorithm 4 FindSkeletonNodes-PathBased

```
Input: Subgraph $G_{st}^k$; Source $s$; Target $t$; Hop constraint $k$; Number of skeleton nodes $h$
Output: $V_S$, a set of skeleton nodes, where $|V_S| \leq h$, and the subgraph $G_{st}$
 1: $\{Partial(s,v), \forall v \in V_{st}\}$ = PUSH-PATH($G_{st}^k$);
 2: $\{Partial(v,t), \forall v \in V_{st}\}$ = PUSH-PATH($r(G_{st}^k)$);
 3: for each $v \in V_{st}$ do
 4:     $PCnt(v) = \sum_{(i,c_i) \in Partial(s,v)} \sum_{\substack{(j,c_j) \in Partial(v,t) \\ i+j \leq k}} c_i c_j$
 5: Let $V_S$ be the top-$h$ nodes in $V \backslash \{s,t\}$ with largest (and non-zero) $PCnt(v)$;
 6: return $V_S$;
 7: Procedure PUSH-PATH($G_{st}^k$)
 8:     Initialize $Partial(s,v) = \emptyset, \forall v \in V_{st}$, $Partial_0(s,s) = 1, Partial_0(s,v) = 0, \forall v \in$
    $V \backslash \{s\}$;
 9:     for $l = 0$ to $k-1$ do
10:         for each edge $(u,v) \in E_{st}$ do
11:             $Partial_{l+1}(s,v) += Partial_l(s,u)$;
12:     for each $v \in V_{st}$ do
13:         $Partial(s,v) = \cup_{l=0}^{k} Partial_l(s,v)$;
```

5.2 Finding Skeleton Nodes

The Path Frequency Based Method According to the definition of G_{st}^k, which is the union of all k-hop paths from s to t, a natural importance measure for a node $v \in V_{st}^k$ is the number of paths that go through v. We denote it as $PCnt(v)$. To be precise, we have

$$PCnt(v) = \sum_{p: s \to t, p \in G_{st}^k, |p| \leq k} \mathrm{I}(p \text{ goes through } v), \forall v \in V_{st}, \tag{1}$$

where $\mathrm{I}(*)$ is an indicator variable. Instead of enumerating and checking all paths, which incurs the excessive $O(n^k)$ cost, we propose an algorithm based on the *push* operation which transfers the information from u to v for each edge (u,v) and the observation that $PCnt(v)$ can be computed by counting the number of paths from s to v and v to t, respectively.

Our algorithm, denoted by FindSkeletonNodes-PathBased, takes G_{st}^k, source node s, target node t, hop constraint k, and skeleton node number h as input. It invokes procedure PUSH-PATH to compute all paths from s to v (and from v to t) for each $v \in V_{st}^k$ (Lines 1-2). Since every such path is a fragment of some path from s to t, it is referred to as the *partial path*. For each node v, $Partial(s,v)$ contains a list of $(step, cnt)$ pairs, which indicates that there are totally cnt distinct paths from s to v of length $step$. Once we have $Partial(s,v)$ and $Partial(v,t)$ for each v, we calculate $PCnt(v)$ by the following equation:

$$PCnt(v) \approx \sum_{(i,c_i) \in Partial(s,v)} \sum_{\substack{(j,c_j) \in Partial(v,t) \\ i+j \leq k}} c_i c_j, \forall v \in V_{st}^k. \tag{2}$$

Intuitively, Equation 2 says that the number of k-hop paths that pass v can be approximated by the number of paths from s to v times the number of paths from v to t.

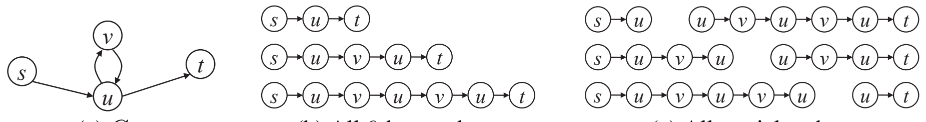

(a) G_{st} (b) All 6-hop paths (c) All partial paths

Fig. 4. Duplicated counting of $s - t$ paths.

Note that we exclude paths longer than k. In fact, the equation gives the exact answer when G_{st}^k does not contain cycles. When the subgraph has cycles, Equation 2 computes an upper bound of $PCnt(v)$. We illustrate it by an example as in Fig. 4. Consider the subgraph in Fig. 4(a), while we set $k = 6$. Their are totally three 6-hop paths from s to t (Fig. 4(b)). Similarly, we can compute all 6-hop paths from s to v and from v to t, as shown in Fig. 4(c). If we concatenate these partial paths together (and eliminate paths longer than 6), the second and third path Fig. 4(b) is counted twice and three times, respectively.

Unfortunately, we are unable to eliminate duplicated counting of paths unless we can enumerate all $s - t$ paths, which is infeasible for sizable graphs. However, since k is usually small in practice, which limits the repetitions in a cycle for a k-hop path, the over estimation of path count only has a minor effect. Therefore, our approximation achieves a good balance between efficiency and effectiveness.

Now we describe the procedure to compute $Partial(s, v)$ (and $Partial(v, t)$) for each node v. Take $Partial(s, v)$ as an example. Denote by $Partial_l(s, v)$ the number of path from s to v of length l, the following lemma holds. The equation for $Partial_l(v, t)$ can be defined analogously.

Lemma 1.

$$Partial_l(s,v) = \begin{cases} 1, & \text{if } v = s \text{ and } l = 0, \\ 0, & \text{if } v \neq s \text{ and } l = 0, \\ \sum_{u \in I(v)} Partial_{l-1}(s,u), & \text{otherwise.} \end{cases} \tag{3}$$

Proof. For any node $v \neq s$, a path from s to v of length l can be decomposed into the sub-path from s to u and edge (u, v), where u denotes the predecessor of v in p. The length of the sub-path is exactly $l - 1$. Besides, for any $u, u' \in I(v)$ and $u \neq u'$, the path to v either comes from u or u', and our lemma follows.

The implementation of Equation 3 is shown in procedure PUSH-PATH. We initialize $Partial(s, v)$ in Line 8, and then proceed on k iterations. During each iteration l, we check every edge $(u, v) \in E_{st}$, and push $Partial_l(s, u)$ to $Partial_{l+1}(s, v)$ (Lines 9-11). For the computation of $Partial(v, t)$, we invoke PUSH-PATH on the reverse graph of G_{st}^k (denoted as $r(G_{st}^k)$). Notice that for practical efficiency, in each iteration l we only record a set of nodes with $Partial_l(s, v) > 0$, and conduct push operation from these nodes. The following lemma and theorem are easily derived.

Lemma 2. *The time and space complexity of procedure PUSH-PATH is $O(k(n + m))$ and $O(kn + m)$, respectively.*

Proof. Since in each iteration, each node and each edge is processed at most once, so the complexity is bounded by $O(|G^k_{st}|) = O(|V^k_{st}| + |E^k_{st}|)$, and again bounded by $O(n + m)$ because G_{st} is a subgraph of G. The algorithm has exactly k iterations, thus the time complexity is $O(k(n + m))$. For the space usage of PUSH-PATH, note that storing $Partial(s, v)$ needs $O(k)$ space. We need extra $O(|G^k_{st}|)$ space for G^k_{st}. In total, the space cost is bounded by $O(kn + m)$.

Theorem 4. *The time and space complexity of FindSkeletonNodes is $O(k^2n + km))$ and $O(kn + m)$, respectively.*

Proof. Algorithm FindSkeletonNodes invokes PUSH-PATH twice, which costs $O(k(n+m))$ time. Since both $Partial(s, v)$ and $Partial(v, t)$ may contain $O(k)$ items, computing $PCnt(v)$ for each v is $O(k^2)$. Consequently, the total computation cost is bounded by $O(k(n + m)) + O(k^2n)$ = $O(k^2n + km))$. Since each node (resp. each edge) needs $O(k)$ (resp. $O(1)$) space cost, the space complexity is $O(kn + m)$.

In practice, we always use small k, e.g., $k \leq 10$, and the algorithm has near-linear time and space complexity.

The Walking Probability Based Method The path frequency based definition of node importance is intuitive, but suffers from a few deficiencies owing to the graph structure inside and outside G^k_{st}. First, the path frequency based measure does not consider the path length, which is an indication of the closeness of the relation. Second, path frequency is vulnerable to malicious tampering of the graph structure. Take Fig. 2(a) as an example, by building more shell companies (i.e., b_i), more $s - t$ paths go through a and c. Third, node d is a hot point but contribute few to the $s - t$ relation, indicating that we should also consider the graph structure of the whole graph when choosing skeleton nodes. Lastly, as previously discussed, the path-based measure is also vulnerable to cycles.

Inspired by the Random Walk with Restart [20] and Personalized PageRank [19], we propose a walking probability based measure for node importance, which alleviates all drawbacks above. We only need a few modification of FindSkeletonNodes-PathBased and PUSH-PATH. Briefly speaking, instead of transfer the information of path frequency from u to v along each edge (u, v), we transfer probability instead. We first define the walking probability as follows.

Definition 6 (Random walk). *A random walk from u is defined as (1) for each step, with probability α the walk stops; (2) with $1 - \alpha$ probability, u randomly chooses an out-neighbor v and proceeds to it. Here α is a decay factor in $(0, 1)$.*

Definition 7 (Walking probability). *The probability of node u walks to v, denoted as $Pr(s, v)$, is defined as $Pr(s, v) = \cup_{l=0}^{k} Pr_l(s, v), \forall v \in V$, where*

$$Pr_l(s,v) = \begin{cases} 1, & \text{if } v = s \text{ and } l = 0, \\ 0, & \text{if } v \neq s \text{ and } l = 0, \\ \sum_{u \in I(v)} \alpha \cdot \frac{Pr_{l-1}(s,u)}{|O(u)|}, & \text{otherwise.} \end{cases} \quad (4)$$

It can be proved by induction that $Pr(s, v)$ is exactly the probability of a random walk from s terminating at v. We omit the details for space constraint. By substituting the path frequency measure by the probability based one, we denote the corresponding procedures as FindSkeletonNodes-ProbBased and PUSH-PROB, respectively.

5.3 Summarized Graph Construction

After we have determined the skeleton node set V_S, we contract G_{st} accordingly. The procedure is rather straightforward: for each $v \in V_S$, we conduct any off-the-shelf local clustering algorithm, and contract each cluster to a super-node. Two super-nodes have connecting edges if some nodes in the corresponding cluster are connected or the walking probability between the super-nodes is above some threshold. For example, if we employ personalized PageRank for local clustering and estimate the PPR values via a limited number of random walks, a good tradeoff between efficiency and effectiveness can be fulfilled.

Finally, we conclude with the following Theorem, which states the complexity of KHGS.

Theorem 5. *The time and space complexity of KHGS is $O(h(k^2n + km))$ and $O(hn + m)$, respectively, and is near-linear when k and h can be viewed as constants.*

6 Experiments

In this section, we evaluate both the efficiency of our k-hop subgraph algorithm against the baseline algorithm, as well as the performance of the summarization algorithms.

6.1 Experimental Settings

Dataset Details We employ four large directed dataset, i.e., Web-Google (WG) ($n = 875,713, m = 5,105,039$), In-2004 (IN) ($n = 1,382,908, m = 16,917,053$), Soc-LiveJournal (LJ) ($n = 4,847,571, m = 68,475,391$), and IT-2004 (IT) ($n = 41,291,594, m = 1,150,725,436$). All datasets are obtained from public sources [1,2].

Methods For the k-hop subgraph queries, we compare the baseline algorithm (Alg. 1) and KHSQ (Alg.2) in terms of efficiency. For the graph summarization problem, we report the query time of KHGS (Alg.3), and consider skeleton node selection via both path frequency based and walking probability based methods.

Environments We randomly generate $1,000$ $s - t$ pairs, and vary k from 3 to 6. We guarantee that t can be reached from s within k hops. All experiments are conducted on a machine with a 2.6GHz CPU and 64GB memory.

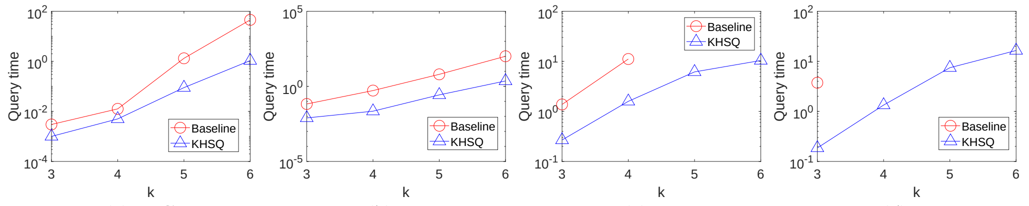
6.2 Efficiency

Table 1 shows the query time of the baseline algorithm and KHSQ. Since the k-DFS procedure is extremely slow for large k, we set $k = 4$. Symbol '-' indicates that for some query the time cost exceeds 1,000 seconds. KHSQ is significantly faster than the baseline, e.g., nearly an order of magnitude faster on LJ. Moreover, for the largest dataset IT, the baseline method fails to answer the query even for $k = 4$. Fig. 5 demonstrate the query speed of Baseline and KHSQ varying k.

In the following, we compare the result size of path enumeration and subgraph query, followed by the query time evaluation of our summarization algorithms.

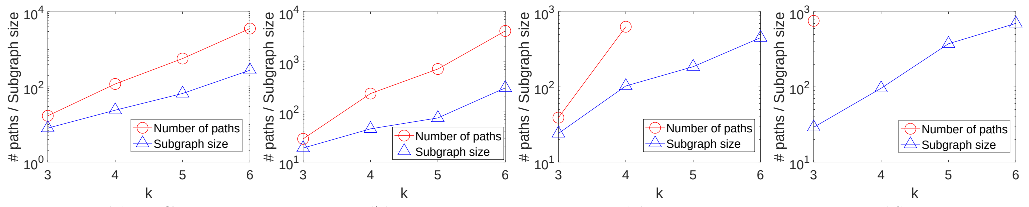
Table 1. Query time (sec) of Baseline and KHSQ ($k = 4$).

Method	Dataset			
	WG	IN	LJ	IT
Baseline	0.013	0.52	11.12	-
KHSQ	0.005	0.022	1.59	1.34



(a) WG (b) IN (c) LJ (d) IT

Fig. 5. Query time (sec) of Baseline and KHSQ, varying k



(a) WG (b) IN (c) LJ (d) IT

Fig. 6. Result size of path enumeration and subgraph query

Table 2. Query time (sec) of KHGS.

Method	Dataset			
	WG	IN	LJ	IT
KHGS (Path-based)	0.43	1.28	76.81	91.26
KHGS (Probability-based)	0.52	1.42	83.57	97.71

Result Size We conduct k-hop path enumeration and subgraph query on four datasets and very k from 3 to 6. The result is shown in Fig. 6. As k increases, the number of k-hop paths explodes, whereas the size of k-hop subgraph is still limited.

Query Time of KHGS We also evaluate the query efficiency of our KHGS algorithm. For all datasets, we fix the number of skeleton nodes h as 8, hop constraint $k = 5$, and $\alpha = 0.6$. Note that the complexity of KHGS is linear in h and quadratic in k. Hence their values do not have a major effect on the query efficiency. The results are shown in Table 2.

7 Conclusion

In this paper, we propose the problem of k-hop $s - t$ subgraph query and a traversal-based algorithm that answers the query in $O(n+m)$ time, which is worst-case optimal. We further introduce the notion of hop-constrained $s - t$ graph summarization, which computes a skeleton graph of the $s - t$ subgraph and provides a better understanding of the underlying structure of the $s - t$ relation. Our proposed algorithms are based on skeleton node selection with various strategies and graph traversal. Extensive experiments demonstrate that our proposed queries better reflect the $s - t$ relation compared to existing queries, while our algorithms are highly efficient even on massive graphs.

Acknowledgements. This work was supported by The National Key Research and Development Program of China under grant 2018YFB1003504, NSFC (No. 61932001), and Peking University Medicine Seed Fund for Interdisciplinary Research supported by the Fundamental Research Funds for the Central Universities (No. BMU2018MI015). This work was also supported by Beijing Academy of Artificial Intelligence (BAAI). Lei Zou is the corresponding author.

References

1. `http://konect.uni-koblenz.de/`
2. `http://law.di.unimi.it/webdata/`
3. Abraham, I., Delling, D., Goldberg, A.V., Werneck, R.F.: Hierarchical hub labelings for shortest paths pp. 24–35 (2012)
4. Bast, H., Funke, S., Sanders, P., Schultes, D.: Fast routing in road networks with transit nodes. Science **316**(5824), 566–566 (2007)
5. Bauer, R., Delling, D., Sanders, P., Schieferdecker, D., Schultes, D., Wagner, D.: Combining hierarchical and goal-directed speed-up techniques for dijkstra's algorithm. ACM Journal of Experimental Algorithmics **15**(2.3) (2010)
6. Chang, L., Lin, X., Qin, L., Yu, J.X., Pei, J.: Efficiently computing top-k shortest path join. In: EDBT 2015-18th International Conference on Extending Database Technology, Proceedings (2015)
7. Cheng, J., Shang, Z., Cheng, H., Wang, H., Yu, J.X.: K-reach: who is in your small world. Proceedings of the VLDB Endowment **5**(11), 1292–1303 (2012)
8. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Robust exact distance queries on massive networks. Microsoft Research, USA, Tech. Rep **2** (2014)
9. Dunne, C., Shneiderman, B.: Motif simplification: improving network visualization readability with fan, connector, and clique glyphs. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 3247–3256. ACM (2013)
10. Eppstein, D.: Finding the k shortest paths. SIAM Journal on computing **28**(2), 652–673 (1998)
11. Geisberger, R., Sanders, P., Schultes, D., Delling, D.: Contraction hierarchies: Faster and simpler hierarchical routing in road networks. In: International Workshop on Experimental and Efficient Algorithms. pp. 319–333. Springer (2008)
12. Goldberg, A.V., Harrelson, C.: Computing the shortest path: A search meets graph theory. In: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms. pp. 156–165. Society for Industrial and Applied Mathematics (2005)
13. Grossi, R., Marino, A., Versari, L.: Efficient algorithms for listing k disjoint st -paths in graphs pp. 544–557 (2018)
14. Jiang, M., Fu, A.W., Wong, R.C., Xu, Y.: Hop doubling label indexing for point-to-point distance querying on scale-free networks. very large data bases **7**(12), 1203–1214 (2014)
15. Jiménez, V.M., Marzal, A.: Computing the k shortest paths: A new algorithm and an experimental comparison. In: International Workshop on Algorithm Engineering. pp. 15–29. Springer (1999)

16. Jin, R., Hong, H., Wang, H., Ruan, N., Xiang, Y.: Computing label-constraint reachability in graph databases. In: Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. pp. 123–134. ACM (2010)
17. LeFevre, K., Terzi, E.: Grass: Graph structure summarization. In: Proceedings of the 2010 SIAM International Conference on Data Mining. pp. 454–465. SIAM (2010)
18. Martins, E.Q., Pascoal, M.M.: A new implementation of yens ranking loopless paths algorithm. Quarterly Journal of the Belgian, French and Italian Operations Research Societies **1**(2), 121–133 (2003)
19. Page, L., Brin, S., Motwani, R., Winograd, T.: The pagerank citation ranking: Bringing order to the web. Tech. rep., Stanford InfoLab (1999)
20. Pan, J.Y., Yang, H.J., Faloutsos, C., Duygulu, P.: Automatic multimedia cross-modal correlation discovery. In: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 653–658. ACM (2004)
21. Peng, Y., Zhang, Y., Lin, X., Zhang, W., Qin, L., Zhou, J.: Hop-constrained st simple path enumeration: Towards bridging theory and practice. Proc. VLDB Endow. **13**(4), 463–476 (2019)
22. Purohit, M., Prakash, B.A., Kang, C., Zhang, Y., Subrahmanian, V.: Fast influence-based coarsening for large networks. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 1296–1305. ACM (2014)
23. Qiu, X., Cen, W., Qian, Z., Peng, Y., Zhang, Y., Lin, X., Zhou, J.: Real-time constrained cycle detection in large dynamic graphs. Proceedings of the VLDB Endowment **11**(12), 1876–1888 (2018)
24. Su, J., Zhu, Q., Wei, H., Yu, J.X.: Reachability querying: can it be even faster? IEEE Transactions on Knowledge and Data Engineering **29**(3), 683–697 (2016)
25. Tang, X., Chen, Z., Zhang, H., Liu, X., Shi, Y., Shahzadi, A.: An optimized labeling scheme for reachability queries. Comput Materials Cont **55**(2), 267–283 (2018)
26. Tian, Y., Hankins, R.A., Patel, J.M.: Efficient aggregation for graph summarization. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 567–580. ACM (2008)
27. Toivonen, H., Zhou, F., Hartikainen, A., Hinkka, A.: Compression of weighted graphs. In: Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 965–973. ACM (2011)
28. Wei, H., Yu, J.X., Lu, C., Jin, R.: Reachability querying: an independent permutation labeling approach. The VLDB JournalThe International Journal on Very Large Data Bases **27**(1), 1–26 (2018)
29. Yen, J.Y.: Finding the k shortest loopless paths in a network. management Science **17**(11), 712–716 (1971)
30. Zhu, A.D., Lin, W., Wang, S., Xiao, X.: Reachability queries on large dynamic graphs: a total order approach. In: Proceedings of the 2014 ACM SIGMOD international conference on Management of data. pp. 1323–1334. ACM (2014)