Querying Minimal Steiner Maximum-Connected Subgraphs in Large Graphs

Jiafeng Hu Xiaowei Wu Reynold Cheng Siqiang Luo Yixiang Fang

Department of Computer Science, The University of Hong Kong
{jhu, xwwu, ckcheng, sqluo, yxfang}@cs.hku.hk

ABSTRACT

Given a graph G and a set Q of query nodes, we examine the *Steiner Maximum-Connected Subgraph* (SMCS). The SMCS, or G's induced subgraph that contains Q with the largest connectivity, can be useful for customer prediction, product promotion, and team assembling. Despite its importance, the SMCS problem has only been recently studied. Existing solutions evaluate the *maximum SMCS*, whose number of nodes is the largest among all the SMCSs of Q. However, the maximum SMCS, which may contain a lot of nodes, can be difficult to interpret. In this paper, we investigate the *minimal SMCS*, which is the minimal subgraph of G with the maximum connectivity containing Q. The minimal SMCS contains much fewer nodes than its maximum counterpart, and is thus easier to be understood. However, the minimal SMCS can be costly to evaluate. We thus propose efficient Expand-Refine algorithms, as well as their approximate versions with accuracy guarantees. Extensive experiments on six large real graph datasets validate the effectiveness and efficiency of our approaches.

1. INTRODUCTION

Graphs are prevalent in various domains, such as social science, e-commerce, and biology. Given a graph G and a set Q of nodes, we study the *Steiner Maximum-Connected Subgraph* (or *SMCS*), a subgraph of G with the maximum connectivity that contains Q. The SMCS can be used in customer prediction, community search, product promotion, and team assembling [6]. In a social network (e.g., Facebook), given a set Q of nodes denoting social network users, its SMCS represents a group of people with similar interest. The members of the SMCS found can then be considered for product recommendation. As another example, in a Protein-Protein-Interaction (PPI) network [3], the SMCS can be used to discover a subgraph connecting a given set Q of protein nodes; the protein nodes appearing in the SMCS can have a close relationship. In a bibliographic network (e.g., DBLP), the SMCS can be used to look for research communities, in order to facilitate collaboration. Figure 1 illustrates an SMCS for the set Q={"Michael Stonebraker", "Samuel Madden", "Daniel J. Abadi", "Jennie Duggan"}, extracted from the DBLP. This SMCS illustrates the researchers who are related to those specified in Q.

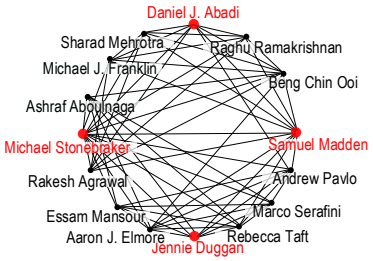


Figure 1: A minimal SMCS of the DBLP.

The notion of the SMCS has only been recently studied. In particular, Chang et al. [6] investigated a variant of the SMCS (or known as the *Steiner Maximum-Connected Component* (SMCC) in [6]), which is the *maximum SMCS* whose number of nodes is the largest among all the possible SMCSs. We have tested the solution provided by [6] on some datasets. We found that the maximum SMCS has a high cohesiveness (because its *connectivity*, or the smallest number of edges whose removal disconnects it, is maximized). Unfortunately, the maximum SMCS is often extremely large and complex. On the DBLP dataset that contains $803K$ nodes and $3.2M$ edges, the average number of nodes of a maximum SMCS is over $400K$. This not only hinders the analysis of the SMCS structure, but also makes it difficult to be used in real situations. Suppose that a user wants to set up an academic conference. She has a small budget to invite a few renowned scholars and their related researchers. To decide the invitation list, the user may issue a maximum SMCS query, with Q containing the names of several researchers, on DBLP. She can then contact the researchers (or graph nodes) that appear in the SMCS. However, if the maximum SMCS is very large, the user can have a hard time to figure out the appropriate participants. Is it possible to get a *smaller* SMCS, while maintaining the maximum connectivity?

In this paper, we examine the discovery of an SMCS that has a small number of nodes. One way is to evaluate the *minimum SMCS*, whose number of nodes is the smallest among all the possible SMCSs. However, as we will discuss in Section 3.2, finding the minimum SMCS is NP-hard. Furthermore, it is NP-hard to get an approximate minimum SMCS with any constant ratio. Thus, any attempt to ob-

CIKM'16 , October 24-28, 2016, Indianapolis, IN, USA

DOI: http://dx.doi.org/10.1145/2983323.2983748

tain a minimum SMCS or its approximate version appears to be a futile exercise. We study another version of SMCS, called the *minimal SMCS*, which is essentially an SMCS of Q (denoted as G'), such that any subgraph of G' containing Q is *not* an SMCS of Q. While the minimal SMCS is still challenging to find, we show that it can be derived in polynomial time. To our understanding, the evaluation of the minimum and minimal SMCS's have not been studied before.

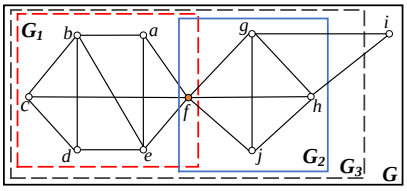


Figure 2: The maximum SMCS (G_3), minimum SMCS (G_2), and minimal SMCSs (G_1 and G_2).

Example 1.1 *Figure 2 illustrates three SMCSs, namely, G_1, G_2, and G_3, for a graph G and query node set $Q = \{f\}$. All these SMCSs have a maximum connectivity of 3, i.e., at least 3 edges have to be removed in order to disconnect them. Here, G_3 is the maximum SMCS, since it has the largest number of nodes, while G_2 is the minimum one. Both G_1 and G_2 are minimal SMCSs.*

Because obtaining the minimum SMCS is computationally intractable, we study the efficient retrieval of the minimal SMCS. One simple way is to first adopt the solution in [6] to compute the maximum SMCS G', and then iteratively refine G' to ensure its minimality. While this solution is simple, it has a high complexity, since the cost of testing the minimality of an SMCS is high (Section 4.2.1). Moreover, due to the huge size of the maximum SMCS, it is extremely slow in our experiments (Section 6).

Our Contributions. We have designed a minimal SMCS solution called the *Expand-Refine* framework. In the *Expand* step, through local expansion of nodes starting from nodes in Q, we obtain a subgraph G' of G, which satisfies the requirement of an SMCS. Intuitively, we obtain G' by exploring the neighboring nodes of Q until we obtain an SMCS. In the *Refine* step, we devise an efficient algorithm that removes nodes based on the dependence of nodes on their minimal SMCSs.

We further improve the efficiency of solutions by relaxing the constraint from two perspectives, namely connectivity and minimality. In the *Expand* step, we bound the expansion space (to relax the connectivity); in the *Refine* step, we develop an approximation solution with accuracy guarantees (to relax the minimality).

We have performed a detailed evaluation of our algorithms on six large real graph datasets. Our experimental results confirm our claim that the minimal SMCS has a higher "quality" than the maximum SMCS, in terms of fewer nodes and higher edge density. The efficiency of our new minimal SMCS solutions addresses several orders of magnitude improvement over basic solutions.

Organization. We review the related work in Section 2. Section 3 formulates the SMCS and analyses the minimum SMCS problem. Section 4 discusses our Expand-Refine solutions. In Section 5, we present our relaxation strategies to further improve the efficiency. We report our results in Section 6. Section 7 concludes.

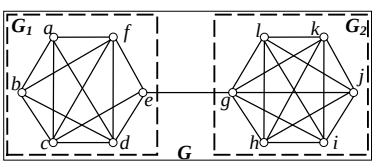
2. RELATED WORK

Our work is related to topics of *connectivity*, *community search*, and *cohesive subgraph detection*.

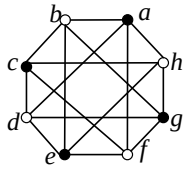
Connectivity. The SMCS is a subgraph of G with the maximum *edge-connectivity* (called *connectivity* here). The connectivity of a graph G is the minimum number of edges whose removal disconnects G [17]. Connectivity has been studied in a wide range of graph-related problems, including network reliability [13], VLSI chip design [22], transportation planning [5], social networks [30], computational biology [27], and cohesive subgraph detection [1, 8]. However, it has only been recently used to facilitate the search of cohesive subgraphs for a given set of nodes (or SMCS) [6]. In this paper, we study the SMCS problem extensively.

Community Search. Given a set Q of nodes in a graph G, the community search problem aims at finding the subgraphs of G that contains Q. For this problem, various goodness metrics have been proposed, including minimum degree [26, 15, 4], trussness [18, 19](the minimum support of an edge in the subgraph, where the support of an edge is the number of triangles containing it), α-adjacency-γ-quasi-k-clique [14], query biased edge density [31], and attributed community [16]. These measures are fundamentally different from connectivity, and so their solutions cannot be used to obtain the SMCS.

Moreover, as mentioned in [1, 20], compared with minimum degree and trussness, connectivity is a better cohesiveness metric. In particular, the minimum degree only restricts degrees of nodes in subgraphs without any structure constraint [1]. In Fig. 3a, with query $Q = \{a, e\}$, the whole graph G will be returned under the minimum degree metric (where the minimum degree is maximized). However, under the connectivity metric, a better subgraph G_1 in which all nodes are highly connected will be returned, since the nodes in G_2 are far away from query nodes. As for the trussness measure, it can be too restrictive on the triangle structure, which is a local concept, whereas connectivity is more global [1]. Notice that there is no triangle in bipartite graphs, e.g., paper-author graphs, online dating graphs, or product-purchaser graphs. For these graphs, it is better to use connectivity as a goodness metric, since no cohesive subgraphs with trussness can be found. For example, in Fig. 3b, with query $Q = \{c\}$, no subgraph will be returned under trussness metric, because there is no triangle in the subgraph. On the other hand, the whole graph (with connectivity equal to 4) is returned under the connectivity metric. Hence, in this paper, we use connectivity for community search, and develop solutions for obtaining the minimal SMCS.



(a)



(b)

Figure 3: (a) An ill-connected graph that minimum degree fails to separate; (b) A well-connected bipartite graph that cannot be found with trussness [1].

Cohesive Subgraph Detection. In recent years, there has been a lot of work about the retrieval of cohesive subgraphs from a large graph. Different kinds of cohesive subgraphs have been studied, such as the maximal clique [7, 10],

quasi-clique [32], k-plex [29], k-core [25, 9, 21], k-truss [12, 28], and locally densest subgraph [24]. However, these solutions are inherently different from our problem, in which they are not query-dependent (i.e., the specification of Q is not required). Hence, their techniques are inapplicable to compute minimal SMCS.

3. THE SMCS PROBLEM

We will describe the graph model and three types of SMCSs in Section 3.1. We study the intractability problem of the minimum SMCS in Section 3.2.

3.1 Connectivity and SMCS

Given an undirected graph G, let $V(G)$ and $E(G)$ be its sets of nodes and edges respectively. We use $G[S] = (S, E[S])$ to denote the subgraph of G induced by node set $S \subseteq V(G)$, where $E[S] = \{(u, v) \in E(G) : u, v \in S\}$. Let $N(u)$ be the set of neighbors of u in G. We denote by $G \backslash u$ the graph obtained by removing node u from $V(G)$. We use *component* to refer to a connected component of G.

Definition 3.1 (Connectivity) *The connectivity (or edge-connectivity in [17]) $\lambda(u, v)$ between two distinct nodes u and v in $V(G)$ is the minimum number of edges whose removal disconnects u and v. The connectivity of the graph $\lambda(G) = \min_{u,v \in V(G)} \lambda(u, v)$ is the minimum connectivity between any two distinct nodes in G (i.e., the smallest number of edges whose removal disconnects G).*

Definition 3.2 (k-Component) *A subgraph g of G is a k-component (or k-edge connected component in [1, 8]) if 1) $\lambda(g) \geq k$; and 2) the connectivity of any super-graph of g in G is less than k.*

Given a set of query nodes $Q \subseteq V$, we use G_Q to denote a node-induced subgraph of G containing Q. We define the SMCS as follows.

Definition 3.3 (SMCS) *The Steiner Maximum-Connected Subgraph (SMCS) is a subgraph G_Q of G such that the connectivity $\lambda(G_Q)$ of G_Q is maximized.*

Let $sc(Q)$ be the connectivity of any SMCS of Q. In [6], $sc(Q)$ is called the *Steiner-connectivity* of Q. When $Q = \{u\}$, we use $sc(u)$ to denote $sc(\{u\})$.

Definition 3.4 (Maximum SMCS) *An SMCS G_Q of Q such that the number of nodes in G_Q is maximized.*

The maximum SMCS is also called the *Steiner Maximum-Connected Component* (SMCC) in [6]. In our experiments, the maximum SMCS suffers from having a huge size and low edge density. Hence, we study the problems of finding other SMCS alternatives as follows.

Definition 3.5 (Minimum SMCS) *An SMCS G_Q of Q such that the number of nodes in G_Q is minimized.*

Definition 3.6 (Minimal SMCS) *An SMCS G_Q of Q such that any proper induced subgraph of G_Q containing Q is not an SMCS of Q.*

Note that a minimum SMCS is also a minimal SMCS. Figure 2 shows these three kinds of SMCSs. Next, we discuss the problem of finding the minimum SMCS.

3.2 Intractability of minimum SMCS

It is easy to observe that the minimum SMCS problem is APX-hard since it is a generalization of the `Steiner tree` problem: given any subset of nodes S in $G(V, E)$, a minimum subgraph spanning all nodes in S can be found by computing a minimum SMCS of $S \cup \{u\}$ in $G(V \cup \{u\}, E \cup \{(s, u)\})$, where $s \in S$ and $u \notin V$. Note that although the objectives are slightly different (one aims at minimizing the number of edges while the other aims at minimizing the number of nodes), we can create dummy nodes within each edge to make the two objectives arbitrarily close. Since the `Steiner tree` problem is APX-hard, the minimum SMCS is also APX-hard. We further show in the following that the problem does not admit any constant approximation ratio, even when restricted to the case when $|Q| = 1$. The reduction from `vertex cover` problem is a modification of the hardness proof of the MSMD_3 problem in [2].

Theorem 3.1 (Inapproximability) *Unless P=NP, there does not exist any polynomial-time algorithm that approximates the minimum SMCS problem within any constant ratio.*

Proof. We show that there exists an instance of minimum SMCS problem with one query node such that is NP-hard to approximate within any constant ratio, which proves that the general minimum SMCS problem does not admit any polynomial-time constant approximation algorithm unless P=NP. We first give an APX-hard instance.

Let H be an instance of `vertex cover` with n_H nodes and m_H edges that does not admit any polynomial-time approximation scheme (PTAS). W.l.o.g., assume $m_H = 3 \cdot 2^h = O(n_H)$ for some integer h and the minimum degree of H is at least 3. We create an instance H_1 of the minimum SMCS problem as follows. Construct a ternary tree T rooted at r with height $h + 1$ such that every internal node has degree 3. Note that T contains $3 \cdot 2^h$ leaf nodes, called E, each of them corresponds to an edge in H. We then create another copy of E, called F, that forms a Hamiltonian cycle with the previous leaf nodes. We further create n new nodes A such that each of them is connected to a node in F **iff** it is an endpoint of the corresponding edge (refer to Figure 4). Let the minimum SMCS instance be $(H_1, \{r\})$.

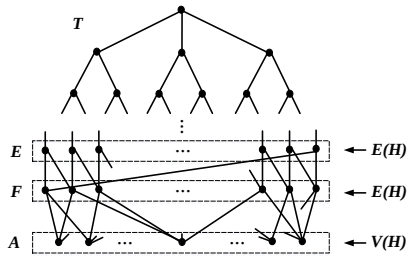


Figure 4: Hard Instance H_1 [2]

Observe that $sc(r) = 3$ and any SMCS of r must contain all nodes in $T \cup F$, but not necessarily all of A. We show that the minimum SMCS of r corresponds to find a minimum $S \subseteq A$ such that every node in F is connected to at least one node in S (hence a vertex cover in H).

Let $S \subseteq A$ be any vertex cover of H. Then the subgraph induced by $T \cup F \cup S$ is an SMCS of r since deleting any two edges from the subgraph cannot disconnect it: since E and F form a cycle, (1) if both edges are chosen from the cycle,

then each node in $S \cup F$ is connected to some node in E, all nodes in E are connected by internal nodes of the ternary tree and hence the graph remain connected; (2) otherwise E and F are still connected by a path P. Since each node not in $E \cup F$ has at least 3 edge-disjoint paths to P, the graph will remain connected. It is easy to observe the other direction: if $T \cup F \cup S$ induces an SMCS of r, then S must be a vertex cover since each node in F has degree 2 in the subgraph induced by $T \cup F$.

Since the `vertex cover` instance H does not admit any PTAS(and $m_H = O(n_H)$), the instance H_1 does not admit any PTAS. Now we proceed to show that there exists an instance of minimum SMCS problem that does not admit any constant approximation ratio by applying the standard error amplification technique.

Given graph H_1, in which the minimum SMCS of $Q = \{r\}$ cannot be approximated within $1 + \epsilon$, we construct H_2 as follows. For each node u (of degree $d(v)$) in H_1, we replace it by a copy of H_1 by connecting each of the $d(v)$ neighbors of u to a distinct internal node $x \in T$ in the copy of H_1. We further replace each x (which is of degree 4 now) by a cycle of length 4 and connect each of its neighbor to a distinct node in the cycle. Let the resulting graph be H_2. Now consider the minimum SMCS of $Q = \{r_2\}$ in H_2, where r_2 is the root of the copy H_1 of the original root r. Still we have $sc(r_2) = 3$. Moreover, for any SMCS of r_2 in H_2, if we extract the subgraph within one copy of H_1 and contract the length 4 cycle we created during the construction, then its minimum degree is at least 3. Hence our previous analysis shows that it must be 3-connected.

Since it is NP-hard to distinguish the existence of a size-k SMCS and a size-$(1+\epsilon)k$ SMCS of Q in H_1, it is now NP-hard to distinguish size-k^2 and size-$(1+\epsilon)^2k^2$ SMCSs (ignoring constant additive terms) in H_2. Hence H_2 does not admit any polynomial-time algorithm that with approximation ratio smaller than $(1+\epsilon)^2$. By repeating the above procedure, we can create instance H_i such that finding the minimum SMCS of r_i is $(1+\epsilon)^i$-inapproximable, which implies that in general, the minimum SMCS problem does not admit any polynomial-time constant approximation algorithm: if there exist a polynomial-time c-approximation algorithm, for some constant $c > 1$, then we would be able to distinguish size-k^i SMCS and size-$(1+\epsilon)^i k^i$ SMCS in H_i, for $i \geq \log_{1+\epsilon} c + 1$. □

Theorem 3.1 states that it is not only intractable to obtain a minimum SMCS, but also hard to get its approximate version in an accurate manner. Hence, we focus on the minimal SMCS, which although may be larger than the minimum SMCS, can be found in polynomial time.

4. PROCESSING MINIMAL SMCS

We now examine how to find a minimal SMCS from G efficiently. Let us first present our *Expand-Refine* framework. As shown in Algorithm 1, it contains three steps. First, we compute the Steiner-connectivity $sc(Q)$ of the query node set Q. We then perform `Expand` (line 2) to generate an SMCS of Q, which serves as a candidate of the minimal SMCS. We then execute `Refine` (line 3) on the SMCS returned by line 2. This operation removes selected nodes and returns a minimal SMCS of Q.

Step 1: Find $sc(Q)$. An efficient algorithm for obtaining $sc(Q)$ has been proposed in [6]. The solution is based on the

Algorithm 1: `Framework`(G, Q)

Input: A graph $G = (V, E)$, and a set Q of query nodes
Output: The node set of a minimal SMCS of Q

1 Compute the Steiner-connectivity $sc(Q)$ of Q; `/* By invoking the method SC-OPT proposed in [6] */`
2 $H \leftarrow$ `Expand`$((V_{sc(Q)}, E_{sc(Q)}), Q, sc(Q))$; `/* Compute an SMCS of Q as a candidate */`
3 $H_Q \leftarrow$ `Refine`$(G[H], Q, sc(Q))$; `/* Refine G[H] to get a minimal SMCS of Q */`
4 **return** H_Q;

connectivity graph. Let us use $sc(u, v)$ to denote $sc(\{u, v\})$. If $(u, v) \in E$, then $sc(u, v)$ is the maximum connectivity of any graph containing edge (u, v).

Definition 4.1 (Connectivity Graph [6]) *Given a graph $G = (V, E)$, the connectivity graph of G is a weighted undirected graph $G_c = (V, E, w)$ with the same set of nodes and edges as G. Each edge (u, v) in G_c has a weight $w(u, v)$ equal to the Steiner-connectivity $sc(u, v)$ of $\{u, v\}$ in G.*

An efficient algorithm was proposed in [6] to compute G_c in $O(\alpha(G) \cdot h \cdot l \cdot |E|)$ time, where $\alpha(G)$ is the "arboricity" of G and is bounded by (usually much smaller than) $\sqrt{|E|}$ [11], and h and l are usually bounded by small constants for real graphs [8]. Based on G_c, [6] developed an $O(|Q|)$ algorithm to compute the Steiner-connectivity $sc(Q)$. For a single node u, its Steiner-connectivity is $sc(u) = \max_{v \in N(u)} sc(u, v)$, i.e., the maximum weight among all edges adjacent to u in G_c.

From now on, we assume that G_c has been computed. Thus, the $sc(u, v)$ and $sc(u)$ values for every edge $(u, v) \in E$ and node $u \in V$ are readily known. In the rest of this section, we will explain the details of `Expand` (Section 4.1) and `Refine` (Section 4.2).

4.1 The Expand Operation (Alg. 1 Step 2)

The goal of this step is to return a candidate for minimal SMCS of Q. In fact, any SMCS of Q can be a candidate, and one simple way is to obtain the maximum SMCS, using the solution provided by [6]. However, as we will show in our experiments, the maximum SMCS computed can be extremely large. If the maximum SMCS of Q is returned in this step, the efficiency of `Refine` (Step 3 of Algorithm 1) can be seriously affected – a huge number of nodes have to be removed before we can get the minimal SMCS of Q. We next present a better method that generates a smaller SMCS of Q efficiently.

To start with, let us present the notion of *layer*.

Definition 4.2 (Layer) *For all $k \geq 1$, let $V_k = \{u \in V : sc(u) \geq k\}$ and $E_k = \{(u, v) \in E : sc(u, v) \geq k\}$. We call $G_k = (V_k, E_k)$ the k-th layer of graph G.*

Given an integer k, G_k can be obtained easily. We first compute E_k, which contains all the edges in G_c whose sc values are higher than k. Then, the set V_k is simply the set of nodes induced by E_k. We have developed a useful lemma for G_k.

Lemma 4.1 *For all $k \geq 1$, each component of the k-th layer $G_k = (V_k, E_k)$ is k-connected.*

PROOF. By definition, we know that for any edge (u, v) in the component G' of (V_k, E_k), there must exists a subgraph G_{uv} of G' containing (u, v) that is k-connected. Hence any two adjacent nodes in G' are k-connected, implying that G' is k-connected. □

Algorithm 2: `Expand`(G_k, Q, k)

Input: A graph $G_k = (V_k, E_k)$, a set Q of query nodes, and the Steiner-connectivity k of Q
Output: The node set of an SMCS of Q

1 $S \leftarrow \emptyset$, $H \leftarrow \emptyset$;
2 Compute a Steiner Tree S from G_k containing Q;
3 **while** $H = \emptyset$ **do**
4 pick $u \in S$ closest to Q such that $N(u) \not\subseteq S$; /* $N(u)$ is the set of neighbors of u in G_k */
5 $S \leftarrow S \cup N(u)$;
6 $H \leftarrow$ `ComputeKECC`($G_k[S], Q, k$); /* Compute a k-component containing Q in $G_k[S]$ */
7 **return** H;

We now describe our *local expansion* strategy, which finds a subgraph of G containing Q that is $sc(Q)$-connected. Let $k = sc(Q)$ be obtained in Step 1 of Algorithm 1. Given the k-th layer G_k=(V_k, E_k) and a set Q of query nodes, Algorithm 2 computes an SMCS of Q by performing a local search on G_k. Particularly, we first form a Steiner tree on the graph G_k to connect all query nodes (line 2). Since the Steiner tree problem is NP-hard, a well-known 2-approximation algorithm [23] is adopted to construct the Steiner tree. We then iteratively expand the candidate node set S by involving the local neighbors of the query nodes in a breadth-first-search manner and invoke `ComputeKECC` to test whether there exists a k-component containing Q in $G[S]$, until a valid SMCS of Q is found (lines 3-6). Here, function `ComputeKECC` returns the node set of that component if it exists, or an empty set otherwise (line 6). To implement this function, we first invoke the best k-component algorithm, called `KECCs-Exact` [8], which returns all the k-components of $G_k[S]$ in $O(h \cdot l \cdot |E|)$ time, where h and l are usually bounded by small constants. Then `ComputeKECC` returns the k-component that contains Q. For convenience, we use $\mathsf{M} = O(h \cdot l \cdot |E|)$ to denote the time complexity of `ComputeKECC`.

Algorithm 2 always returns an SMCS of Q, since in the worst case, the maximum SMCS will be returned. In our experiments, the number of nodes in SMCSs returned is much smaller than that of their maximum counterparts.

4.2 The Refine Operation (Alg. 1 Step 3)

After obtaining an SMCS $G[H]$ of Q with $sc(Q)$=k in Step 2, we need to check its minimality, i.e., any subgraph induced by a proper subset of H is not an SMCS of Q. However, since there are $2^{|H|-|Q|}$ possible induced subgraphs of H containing Q, examining all possible subgraphs is not feasible. To validate the minimality of the SMCS, we first describe a basic refinement algorithm in Section 4.2.1, and then propose a better solution in Section 4.2.2. To further improve the efficiency, we introduce the incremental removal optimization technique in Section 4.2.3.

4.2.1 Basic Refinement

Observe that if $G[H]$ is not minimal, then there must exist a subgraph $G[H_Q]$, where $H_Q \subsetneq H$, $Q \subseteq H_Q$ and $\lambda(G[H_Q]) = k$, i.e., there exists a k-component containing Q if any node in $H \backslash H_Q$ is removed. Based on this intuition, we develop the `Refine-Basic` (Algorithm 3). In lines 1-4, all nodes in $H \backslash Q$ are tested iteratively. If a smaller SMCS is found, we shrink the candidate set and recursively call `Refine-Basic` to find a minimal SMCS (lines 3-4). The node set H of a minimal SMCS will be returned if there

Algorithm 3: `Refine-Basic`($G[H]$, Q, k)

Input: A graph $G[H]$, a set Q of nodes, and the Steiner-connectivity k of Q
Output: The node set of a minimal SMCS of Q

1 **for** *each node* $u \in H \backslash Q$ **do**
2 $H' \leftarrow$ `ComputeKECC`($G[H] \backslash u, Q, k$); /* Compute a k-component containing Q in $G[H] \backslash u$ */
3 **if** $H' \neq \emptyset$ **then**
4 **return** `Refine-Basic`($G[H']$, Q, k);
5 **return** H;

does not exist a k-component containing Q when any node in $H \backslash Q$ is removed (line 5).

Theorem 4.1 *Algorithm 3 returns a minimal SMCS of Q in $O(|H|^2\mathsf{M})$ time.*

Proof. **Correctness:** If $G[H]$ is not minimal, then there exists $H_Q \subsetneq H$, $Q \subseteq H_Q$ and $\lambda(G[H']) = k$. Hence by examining all nodes in $H \backslash Q$, we can find at least one node $u \in H \backslash H_Q$ such that the k-component containing Q after removing u is non-empty, which means that whenever a graph is returned, it must be a minimal SMCS of Q.

Running Time: Since the size of SMCS is reduced in each recursive call, there are at most $|H|$ recursive calls. Between two consecutive recursive calls, each node in $H \backslash Q$ is examined at most once, which takes $O(\mathsf{M})$ time. Hence the total running time is at most $O(|H|^2\mathsf{M})$. □

4.2.2 Advanced Refinement

We now explain how to use the relationship between nodes to speed up the refinement step.

Definition 4.3 (Separable) *Given any SMCS $G[H]$ of a set Q of nodes, node $u \in H \backslash Q$ is* separable *for Q if there exists an $sc(Q)$-component containing Q in $G[H] \backslash u$.*

Lemma 4.2 *Given any SMCS $G[H]$ of a set Q of nodes, if $u \in H \backslash Q$ is non-separable for Q, then any SMCS of Q that is a subgraph of $G[H]$ must contain u.*

Proof. We prove this by contradiction. Let $k = sc(Q)$, and $G[H']$ be the subgraph of $G[H]$ that is an SMCS of Q and does not contain u. Since u is not contained in $G[H']$, when u is deleted from $G[H]$, the k-component containing Q must contain $G[H']$ as a subgraph. Hence u is separable and contradicts the assumption. □

Given an SMCS $G[H]$ of Q, Algorithm 4 computes a minimal SMCS of Q by repeatedly removing separable nodes from the SMCS. We initialize T as the set of nodes whose separabilities are not yet tested (line 1). Then in every round we test the separability of a node $u \in T$ (lines 2-8). If u is non-separable, it will be removed from T (line 6); otherwise it is separable, then we can shrink H to a proper subset $H' \subsetneq H$ that does not contain u and update T (line 8). Note that we can remove at least one node from T in each step and when T is empty, all nodes except those in Q in the current SMCS must be non-separable, which ensures the minimality.

Theorem 4.2 (Minimal-SMCS) *Given an SMCS $G[H]$ of a set Q of nodes,* `Refine`($G[H]$,Q,k) *computes a minimal SMCS of Q that is a subgraph of $G[H]$ in $O(t \cdot \mathsf{M})$ time, where t is the number of iterations ($t < |H|$).*

Algorithm 4: `Refine`($G[H]$, Q, k)

```
1 T ← H\Q;    /* T is the set of all nodes in H whose
  separability is not tested */
2 while T ≠ ∅ do
3 |  pick a node u ∈ T;
4 |  H' ← ComputeKECC(G[H]\u, Q, k);   /* Compute a
  |  k-component containing Q in G[H]\u */
5 |  if H' = ∅ then                    /* u is non-separable */
6 |  |  T ← T\u;
7 |  else                              /* G[H'] is an SMCS of Q */
8 |  |  H ← H', T ← T ∩ H';
9 return H;
```

Proof. First observe that whenever we replace the SMCS $G[H]$ of Q by a subgraph $G[H']$, we have $\lambda(G[H']) = k$, which means that $G[H']$ is also an SMCS of Q. When $G[H]$ is returned by the algorithm, all nodes except those in Q must be non-separable. By Lemma 4.2, we know that $G[H]$ must be a minimal SMCS since any subgraph of $G[H]$ induced by a proper subset of H containing Q must not be k-connected due to the absence of some non-separable node. Since in each iteration, the complexity is $O(\mathsf{M})$, the total running time is $O(t \cdot \mathsf{M})$. □

Given the above algorithms, a minimal SMCS of Q can be computed by calling Algorithm 1 with inputs G and Q (`Framework`(G, Q)).

4.2.3 Incremental Removal Optimization

Note that given an SMCS $G[H]$ of a set Q of query nodes with $sc(Q) = k$, in the refinement step, we identify the separability of a sample node in each iteration. If the sampled node u is separable, then all nodes not contained in the k-component containing Q in $G[H]\backslash u$ are separable. However, in some cases, especially when k is small, every time when a separable node is identified, we can only reduce the size of SMCS by a small constant, which is inefficient when the candidate SMCS is large.

One observation is, for any subset $U \subseteq H\backslash Q$, if there exists a k-component containing Q after removing U from $G[H]$, then we can identify all nodes in U separable immediately. Using this idea, we develop an incremental removal optimization for the refinement algorithm, as shown in Algorithm 5. The key idea is that we try to sample a set of nodes in each iteration. Let i denote the number of nodes to be sampled, initialized to 1 (line 2). After one successful removal during the iteration procedure (a k-component containing Q is found after removing U), the sample size i will be increased by 1 (line 12). When it meets an unsuccessful removal (there does not exist a k-component containing Q after removing U), if the size of U is 1, the only element in U will be removed from T since it is non-separable (line 8); otherwise, the sample size i will be reset to 1 (line 10). Note that in the second case, the nodes in U cannot be labeled as non-separable, even though we know one of them is.

5. IMPROVING PERFORMANCE BY CONSTRAINT RELAXATION

In this section, we focus on improving the efficiency of our `Expand-Refine` algorithm in two ways. First, for the *Expand* operation, we constrain the local search space while relaxing the connectivity. Second, for the *Refine* operation, we propose an approximation algorithm to speed up the refinement procedure with accuracy guarantee.

Algorithm 5: `Refine-Inc`($G[H]$, Q, k)

```
1  T ← H\Q;
2  i ← 1;          /* the number of nodes to be sampled */
3  while T ≠ ∅ do
4  |  i ← min(i, |T|);
5  |  sample i nodes U from T;
6  |  H' ← ComputeKECC(G[H\U], Q, k);   /* Compute a
   |  k-component containing Q in G[H\U] */
7  |  if H' = ∅ and i = 1 then          /* U = {u} */
8  |  |  T ← T\U;
9  |  else if H' = ∅ and i > 1 then     /* some u ∈ U is
   |  non-separable */
10 |  |  i ← 1;
11 |  else                              /* H' ≠ ∅ */
12 |  |  H ← H', T ← T ∩ H', i ← i + 1;
13 return H;
```

5.1 Early Stop in the Expand Step

As described in Section 4.1, in the *Expand* step, we locally expand the Steiner tree S to find an SMCS of Q. However, since the local expansion is a heuristic search strategy, the time cost may be very high in some cases, especially when the graph size is very large. In order to reduce the search space, we use a threshold θ to bound the size of S, so called *early stop*. Specifically, as shown in Algorithm 6, after obtaining the Steiner tree S for the query nodes, we expand the tree S to a graph in a BFS manner until the node size exceeds a threshold θ, i.e., $|S| > \theta$, where θ is empirically tuned. Then we extract a k'-component H containing Q in $G_k[S]$, where $k' \leq k$ is the maximum possible connectivity. Following that, we may get a candidate subgraph with connectivity less than the Steiner-connectivity of Q. However, as we will show in Section 6.3, the connectivity k' of the subgraph returned by Algorithm 6 is very close to the corresponding maximum one in practice.

5.2 Approximation in the Refine Step

Note that to ensure an SMCS graph $G[H]$ of a set Q of nodes is minimal, we need to test separability for each node $u \in H\backslash Q$, which takes $\Omega(|H| \cdot \mathsf{M})$ time in the worst case. For a large k, usually we have a large minimal SMCS, which leads to long processing time. To further improve the efficiency of the refinement procedure, we propose an approximation algorithm which incorporates two extra user-specified parameters into the input, i.e., the approximation ratio r and the failure probability δ. The approximation algorithm stops earlier and outputs with probability at least $(1-\delta)$ an SMCS of Q that is an r-approximation of a minimal SMCS of Q, for any constant $\delta \in (0,1)$ and $r > 1$. Note that an SMCS $G[H]$ of Q is an r-approximation if there exists $H_Q \subseteq H$ such that $|H_Q| \geq \frac{1}{r}|H|$ and $G[H_Q]$ is a minimal SMCS of Q. The failure probability δ is defined as the probability that the approximation ratio is larger than r.

Algorithm 6: `Bounded-Expand`(G_k, Q, k, θ)

Input: A graph $G_k = (V_k, E_k)$, a set Q of query nodes, the Steiner-connectivity k of Q, and a node size threshold θ
Output: The node set of a k'-component containing Q and the connectivity k'

```
1 Compute a Steiner Tree S from G_k containing Q;
2 Expand S by adding the local neighbors of Q in G_k in a
  BFS manner, s.t. |S| ≤ θ;
3 Extract a k'-component H containing Q from S, where
  k' ≤ k is the maximum possible connectivity;
4 return (H, k');
```

Algorithm 7: `Approx-Refine`($G[H]$, Q, k, r, δ)

Input: A graph $G[H]$, a set Q of nodes, the Steiner-connectivity k of Q, and two user-specified parameters r and δ
Output: The node set of an approximated minimal SMCS of Q

1 $T \leftarrow H \backslash Q$;
2 $step \leftarrow 0$; /* the number of non-separable nodes sampled consecutively */
3 **while** $step < \frac{\log \frac{1}{\delta}}{\log r}$ **and** $T \neq \emptyset$ **do**
4 sample uniformly at random a node $u \in T$;
5 $H' \leftarrow$ `ComputeKECC`$(G[H]\backslash u, Q, k)$; /* Compute k-component containing Q in $G[H]\backslash u$ */
6 **if** $H' = \emptyset$ **then** /* u is non-separable */
7 $T \leftarrow T \backslash u$, $step \leftarrow step + 1$;
8 **else**
9 $H \leftarrow H'$, $T \leftarrow T \cap H'$, $step \leftarrow 0$;
10 **return** H;

The pseudocode of the approximation method is shown in Algorithm 7. Different from Algorithm 4, we maintain an extra variable, i.e., the number of non-separable nodes sampled consecutively, denoted as *step*. At first, the value of *step* is initialized to 0 (line 2). In each iteration of the algorithm, we sample a node u from T uniformly at random (line 4) and test its separability (lines 5-9). If u is non-separable (line 6), then it will be removed from T and the value of *step* will be increased by 1 (line 7); otherwise we can shrink H to the k-component containing Q in $G[H]\backslash u$, update T and reset the value of *step* to 0 (line 9). The iteration will be halted when the value of *step* is not less than $\frac{\log \frac{1}{\delta}}{\log r}$ or T is empty. Lemma 5.1 shows that our approximation algorithm outputs an r-approximation of a minimal SMCS of Q in $G[H]$ with probability at least $(1-\delta)$.

Lemma 5.1 (Approximated Minimal SMCS) *Given an SMCS $G[H]$ of a set Q of nodes , for any constant $\delta \in (0,1)$ and $r > 1$, Algorithm 7 returns an r-approximation of a minimal SMCS of Q in $G[H]$ with probability at least $(1-\delta)$.*

Proof. Let $G[H']$ be the SMCS returned by Algorithm 7. Fix any arbitrary minimal SMCS $G[H'_Q]$ of Q in $G[H']$. We prove that the probability that $G[H']$ is not an r-approximation of $G[H'_Q]$, i.e., contains more than $r|H'_Q|$ nodes, is at most δ. When H' is returned, if $T = \emptyset$, then we know that H' contains only non-separable nodes and is a minimal SMCS of Q. Hence we have $T \neq \emptyset$ and $step \geq \frac{\log \frac{1}{\delta}}{\log r}$.

Notice that throughout the whole algorithm, T contains the set of nodes whose separability is not tested in the current SMCS of Q, and the value of *step* is the number of non-separable nodes that are consecutively sampled. By Lemma 4.2, for each non-separable node u, any minimal SMCS of Q, including $G[H'_Q]$, must contain u. Hence in each iteration, all nodes not in H'_Q are contained in T and must be separable. Let x be the number of non-separable nodes that have been identified so far. Note that all those nodes are contained in H'_Q. The probability that a non-separable node is chosen is at most

$$\frac{|H'_Q| - |Q| - x}{|T|} \leq \frac{|H'_Q| - |Q| - x}{|H'| - |Q| - x} < \frac{|H'_Q|}{|H'|} < \frac{1}{r}.$$

Since whenever a non-separable node is chosen, it will be removed from T, the event that a non-separable node is sampled is negatively correlated. Since the probability that a non-separable node is sampled in each iteration is less than $\frac{1}{r}$, the probability of sampling a consecutive number of $\frac{\log \frac{1}{\delta}}{\log r} = \log_r \frac{1}{\delta}$ non-separable nodes is $< (\frac{1}{r})^{\log_r \frac{1}{\delta}} = \delta$. □

Algorithm 8: `Approx-Refine-Inc`($G[H]$, Q, k, r, δ)

1 $T \leftarrow H \backslash Q$, $i \leftarrow 1$, $step \leftarrow 0$;
2 **while** $step < \frac{\log \frac{1}{\delta}}{\log r}$ **and** $T \neq \emptyset$ **do**
3 $i \leftarrow \min(i, |T|)$;
4 sample uniformly at random i nodes U from T;
5 $H' \leftarrow$ `ComputeKECC`$(G[H\backslash U], q, k)$;
6 **if** $H' = \emptyset$ **and** $i = 1$ **then**
7 $T \leftarrow T \backslash U$, $step \leftarrow step + 1$;
8 **else if** $H' = \emptyset$ **and** $i > 1$ **then**
9 $i \leftarrow 1$;
10 **else**
11 $H \leftarrow H'$, $T \leftarrow T \cap H'$, $i \leftarrow i + 1$, $step \leftarrow 0$;
12 **return** H;

As an example, if we set $r = 4$ and require a failure probability $\delta = 0.1\%$, then we can output the current graph whenever we sample consecutive $\frac{\log \frac{1}{\delta}}{\log r} = \frac{\log \frac{1}{0.001}}{\log 4} \approx 4.98 < 5$ non-separable nodes. Note that there is a trade-off between the failure probability δ and the approximation ratio r, i.e., for a fixed number $\lceil \frac{\log \frac{1}{\delta}}{\log r} \rceil$, the failure probability δ increases when r decreases. Since in our analysis, we do not take into consideration the inconsecutive non-separable nodes sampled and the effect that the events of sampling non-separable nodes are negatively correlated, in practical, the approximation ratio and the failure probability should be much smaller (which will be confirmed by our experimental results).

The incremental removal optimization can also be used in the approximation algorithm, as shown in Algorithm 8. The main idea is similar to Algorithm 5. Whenever we sample a separable node, the sample size i is increased by 1 (line 11), otherwise it will be set to 1 (line 9). Since our algorithm stops only when $\lceil \frac{\log \frac{1}{\delta}}{\log r} \rceil$ consecutive non-separable nodes are sampled (which means the sample size is always 1), Lemma 5.1 still holds for Algorithm 8.

6. EXPERIMENTAL RESULTS

Data. We use six large real graph datasets: (1) *ca-CondMat*, or condensed matter collaboration network; (2) *soc-Epinions1*, the who-trusts-whom network of Epinions.com; (3) *DBLP*, a bibliographic network [1]; (4) *wiki-Talk*, a Wikipedia talk (communication) network; (5) *as-Skitter*, the Internet topology; and (6) *uk-2002*, the Web graph within the .uk domain in 2002 [2]. Apart from DBLP and uk-2002, the datasets are downloaded from the Stanford SNAP library [3]. For each dataset, we use its largest connected component as our test graph. Their number of nodes and edges, average degree $\overline{d}$, and the largest Steiner-connectivity sc_{max}, are reported in Table 1.

Queries. The node set Q of a query is randomly generated, based on query size $|Q|$ and the inter-distance l (i.e., the maximum distance between any two nodes in Q). By default, $|Q| = 3$ and $l = 2$. These values are also used in [26, 19]. For testing the effect of the Steiner-connectivity, we use a slightly different query model, as detailed in Section 6.2.

[1] http://dblp.uni-trier.de/xml/
[2] http://law.di.unimi.it/datasets.php
[3] http://snap.stanford.edu/data/

Table 1: Dataset statistics ($\mathbf{K}$=10^3 and $\mathbf{M}$=10^6)

ID	Dataset	#Nodes	#Edges	$\bar{d}$	sc_{max}
D1	ca-CondMat	21**K**	91**K**	8.55	25
D2	soc-Epinions1	75**K**	405**K**	10.69	67
D3	DBLP	803**K**	3.2**M**	8.18	118
D4	wiki-Talk	2.3**M**	4.6**M**	3.90	131
D5	as-Skitter	1.7**M**	11**M**	13.09	111
D6	uk-2002	18**M**	261**M**	28.34	943

Algorithms. We tested several minimal SMCS solutions:

- `Basic`: Based on `Expand-Refine` (Alg. 1), but compute maximum SMCS in line 2, and replace line 3 by `Refine-Basic` (Alg. 3).
- `ER`: Algorithm 1.
- `ER-I`: Algorithm 1 with incremental removal optimization, i.e., replace line 3 by `Refine-Inc` (Alg. 5).
- `ER-I-A`$_\omega$: Approximate `Expand-Refine` with incremental removal optimization, i.e., in Alg. 1, replace line 3 by `Approx-Refine-Inc` (Alg. 8). Here ω=$\lceil \frac{\log \frac{1}{\delta}}{\log r} \rceil$ is the termination threshold, i.e., `ER-I-A`$_\omega$ stops after ω non-separable nodes are sampled.
- `B-ER-I-A`$_\omega$: `ER-I-A`$_\omega$ with bounded local search, i.e., in Alg. 1, replace line 2 by `Bounded-Expand` (Alg. 6) and line 3 by `Approx-Refine-Inc` (Alg. 8). We set the local expansion threshold $\theta = 10000$, which is selected to achieve stable quality and efficiency by testing θ in [500, 20000].

To examine the effectiveness of our solutions, we have also implemented the following algorithms:

- `max-SMCS`: the algorithm proposed in [6], which finds the maximum SMCS of Q.
- `local-SMCS`: this uses local expansion (Alg. 2) to generate an SMCS of Q, without any refinement.

For the parameter ω used in approximation algorithms, we found that ω=3 balances the running time, approximation ratio and failure probability (details in Sec. 6.3). We thus set its default value to 3.

The above algorithms are implemented in C++ and compiled with GNU g++ 4.6.3 with the -O3 optimization. The source codes for computing k-components (`ComputeKECC`), constructing the connectivity graph, and computing the Steiner-connectivity and the maximum SMCS are obtained from the authors in [6, 8]. Our experiments are conducted on a machine with an Intel(R) Xeon(R) CPU@2.6GHz and 96GB memory running Linux.

Next, we examine the effectiveness and efficiency of SMCS solutions in Sections 6.1 and 6.2. We discuss the results for our relaxation solutions in Section 6.3.

6.1 Effectiveness

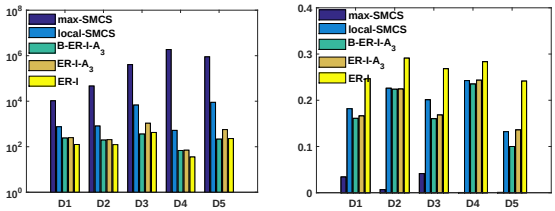
We compare the minimal SMCSs and approximated ones returned by `ER-I`, `ER-I-A`$_3$ and `B-ER-I-A`$_3$ respectively. We also evaluate the quality of `max-SMCS` and `local-SMCS`, according to:

(1) Size. The number of nodes in the result graph.

(2) Edge Density ρ. This measures the density of a graph [17, 24], and is the ratio of the number of edges of a graph g to that of a complete graph with the same set of nodes:

$$\rho(g) = \frac{2 \times |E(g)|}{|V(g)| \times (|V(g)| - 1)}. \quad (1)$$

Exp-1: Quality Evaluation. For each dataset in D1-D5, we randomly select 500 sets of query nodes with the size randomly ranging from 1 to 16 and inter-distance l being 2, and report the average number of nodes size and edge density ρ.



(a) size (b) ρ

Figure 5: Quality evaluation

The results are shown in Figure 5. Note that solutions with local expansion (`local-SMCS`, `B-ER-I-A`$_3$, `ER-I-A`$_3$ and `ER-I`) perform better than `max-SMCS` under all measures. A gigantic number of nodes is returned by `max-SMCS`. Moreover, `ER-I` achieves the highest edge density with the smallest number of nodes. Thus, it is useful to remove nodes from discovered subgraphs by our solution. The number of nodes in the subgraphs returned by `local-SMCS` is around 10 times more than `ER-I`, which shows that using local expansion alone is not enough, and the refinement step is necessary.

Exp-2: DBLP Case Study. We use the query Q={"Michael Stonebraker", "Samuel Madden", "Daniel J. Abadi", "Jennie Duggan"} on the DBLP dataset. Table 2 shows the quality measures for different SMCS methods. We found that `ER-I` returns a small and cohesive 7-connected subgraph (size=15 and ρ=0.58), which is much better than those returned by `max-SMCS` and `local-SMCS`. Figure 1 (in Sec. 1) shows the result returned by `ER-I`. Due to the large sizes of the subgraphs returned by `max-SMCS` and `local-SMCS`, they are not illustrated here. Notice that the results of both `ER-I-A`$_3$ and `B-ER-I-A`$_3$ are all very close to the one returned by `ER-I`.

Table 2: Quality measures of the DBLP case study.

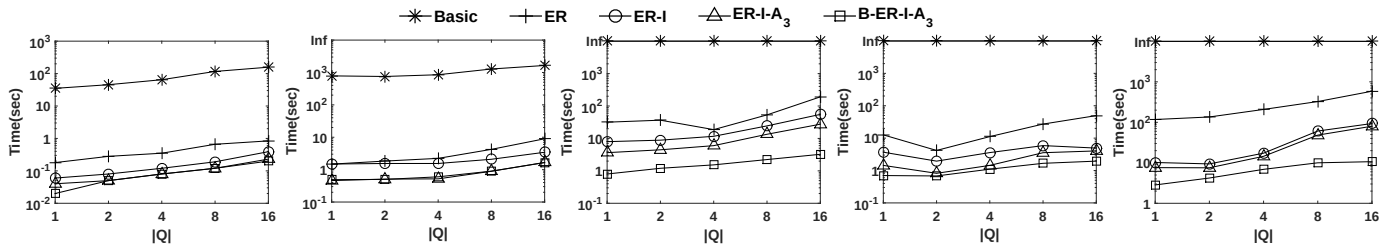
Quality Metric	`max-SMCS`	`local-SMCS`	`B-ER-I-A`$_3$	`ER-I-A`$_3$	`ER-I`
size	171,435	129	17	17	15
ρ	0.0001	0.21	0.54	0.54	0.58

6.2 Efficiency

We now evaluate the efficiency of our algorithms for minimal SMCS queries under different situations. Each experiment is run three times, and the average CPU time is reported in seconds. We treat the running time of a query as infinite (`Inf`) if it exceeds 1 hour.

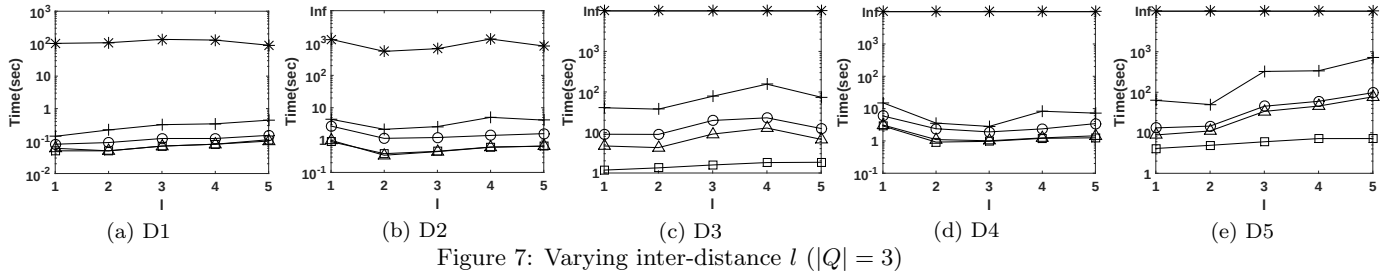
Exp-3: Effect of Queries. In these experiments, we test our approaches using different queries. The reported time is the average time of processing 500 queries.

First, we observe the effect of the query size $|Q|$. We test 5 different $|Q|$ values in {1, 2, 4, 8, 16}. The running time of `ER`, `ER-I`, `ER-I-A`$_3$, `B-ER-I-A`$_3$ and `Basic` on different datasets (D1-D5) is shown in Figure 6. In general, the running time of all algorithms increases with the query size. Since the number of nodes in the candidate generated in the local expansion step increases when $|Q|$ increases, the time cost in both candidate generation step and refinement step increases. Our algorithms (`ER`, `ER-I`, `ER-I-A`$_3$ and `B-ER-I-A`$_3$) outperform the baseline algorithm by several orders of magnitude on all datasets. Moreover, `ER-I` is better than `ER`. For example, on D5 (as-Skitter), `ER-I` is around 10 times faster than `ER`. Thus, the incremental removal optimization improves the performance substantially. The approximation algorithm `ER-I-A`$_3$ further improves the performance by relaxing the minimality; on D3 (DBLP), `ER-I-A`$_3$ is around 2 times faster than `ER-I`. `B-ER-I-A`$_3$ achieves the highest per-



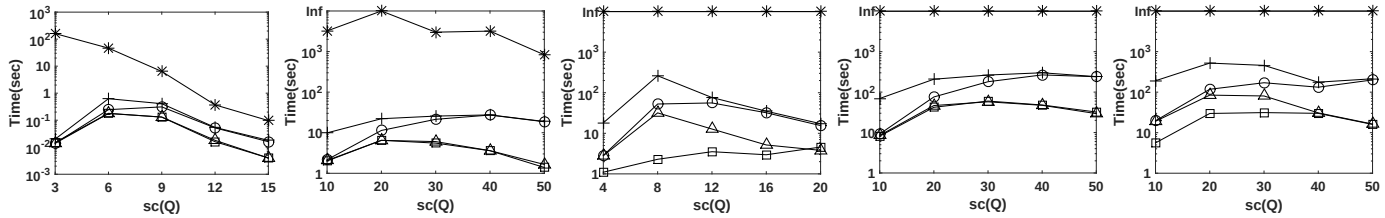
(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

Figure 6: Varying query size $|Q|$ ($l = 2$)



(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

Figure 7: Varying inter-distance l ($|Q| = 3$)



(a) D1 (b) D2 (c) D3 (d) D4 (e) D5

Figure 8: Varying Steiner-connectivity of Q ($|Q| = 3$, $l = 2$)

formance by relaxing both connectivity and minimality; on D3 (DBLP), B-ER-I-A$_3$ is around 10 times faster than ER-I.

We then study the effect of the inter-distance l among query nodes. The running time of ER, ER-I, ER-I-A$_3$, B-ER-I-A$_3$, and Basic on different datasets (D1-D5) by varying the inter-distance l (from 1 to 5) of query is illustrated in Figure 7. Similar to the results obtained by varying the query size, ER, ER-I, ER-I-A$_3$ and B-ER-I-A$_3$ outperform Basic by several orders of magnitude on all datasets. ER-I is still better than ER. For example, on D5 (as-Skitter), ER-I is around 5.7 times faster than ER. Again, B-ER-I-A$_3$ performs the best; on D5, B-ER-I-A$_3$ is around 45 times faster than ER.

We also evaluate the effect of the Steiner-connectivity $sc(Q)$ of Q. For each dataset in D1-D5, we select some representative values of Steiner-connectivity based on its largest Steiner-connectivity. For each value of $sc(Q)$, we randomly select 500 different query sets with $|Q| = 3$ and $l = 2$. The running time of ER, ER-I, ER-I-A$_3$, B-ER-I-A$_3$ and Basic on different datasets by varying the Steiner-connectivity $sc(Q)$ of query is illustrated in Figure 8. Generally, ER, ER-I, ER-I-A$_3$ and B-ER-I-A$_3$ outperform the baseline algorithm by several orders of magnitude on all datasets. When $sc(Q)$ increases, the advantage of the approximation algorithm over others becomes more obvious. For example, on D5 (as-Skitter), when $sc(Q)$=50, ER-I-A$_3$ is around 12.5 times faster than ER-I. When $sc(Q)$ increases, the number of nodes in the result increases, and the approximation algorithm stops earlier while the exact one needs a lot of time to do minimality testing on the result graph. B-ER-I-A$_3$ which combines both connectivity and minimality relaxations still is the fastest.

Table 3: Scalability testing for B-ER-I-A$_3$ on D6 (in seconds)

$\|Q\|$	1	2	4	8	16
Time	13.1	21.3	28.2	32.5	37.9
l	1	2	3	4	5
Time	34.3	41.4	27.5	32.4	29.6
$sc(Q)$	10	20	30	40	50
Time	19.1	17.1	16.1	18.6	18.4

Exp-4: Scalability Testing. We test the scalability of our fastest algorithm (B-ER-I-A$_3$) on D6 (uk-2002) which contains 261 million edges. The reported time is the average time of processing 500 queries. Table 3 shows the running time of B-ER-I-A$_3$ on D6 when varying the query size $|Q|$, inter-distance l and Steiner-connectivity $sc(Q)$. It shows that B-ER-I-A$_3$ has ideal scalability and is quite efficient even for such a large network.

6.3 Results on Constraint Relaxation

In what follows, we will first evaluate the error of connectivity by adding a threshold in the *Expand* step. Then we evaluate our approximation techniques in the *Refine* step and discuss how to set parameters.

Exp-5: Connectivity Relaxation. We evaluate the error of connectivity in B-ER-I-A$_3$ on D3 (DBLP). The average percentage errors of the connectivity (k') of detected minimal SMCSs by B-ER-I-A$_3$ to the one (k) returned by exact algorithms, e.g., ER, are reported in Table 4, where we vary the query size $|Q|$, inter-distance l and Steiner-connectivity $sc(Q)$ as mentioned in Sec. 6.2 ($\%_{\text{error}} = (k - k')/k$). The connectivity of detected minimal SMCSs obtained by B-ER-I-A$_3$ are very close to the exact solutions. Combined with the analysis on efficiency in Sec. 6.2, B-ER-I-A$_3$ balances the efficiency and effectiveness well.

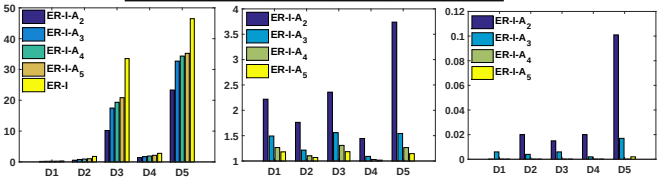
Exp-6: Minimality Relaxation. We evaluate the performance of the approximation algorithm (ER-I-A$_\omega$) by varying

Table 4: Error of connectivity on D3 (DBLP)

$\|Q\|$	1	2	4	8	16
$\%_{\text{error}}$	1%	1%	2%	3%	6%

l	1	2	3	4	5
$\%_{\text{error}}$	1%	1%	2%	2%	2%

$sc(Q)$	4	8	12	16	20
$\%_{\text{error}}$	1%	4%	3%	1%	0%



(a) Query Time(sec) (b) Approx. Ratio (c) Failure Prob.

Figure 9: Minimality approximation performance

the termination threshold ω, from 2 to 5. To investigate the actual approximation ratio and failure probability, for each value of ω from 2 to 5, we select a pair of (r, δ), i.e., (8, 0.02)($\frac{\log \frac{1}{0.02}}{\log 8} \approx 1.88 < 2$), (5, 0.01)($\frac{\log \frac{1}{0.01}}{\log 5} \approx 2.86 < 3$), (6, 0.001)($\frac{\log \frac{1}{0.001}}{\log 6} \approx 3.86 < 4$) and (4, 0.001)($\frac{\log \frac{1}{0.001}}{\log 4} \approx 4.98 < 5$). We randomly select 500 sets of query nodes with size randomly ranging from 1 to 16 and the inter-distance l being 2. The query time, actual approximation ratio and failure probability of `ER-I-A`$_2$, `ER-I-A`$_3$, `ER-I-A`$_4$ and `ER-I-A`$_5$ are shown in Figure 9. In Figure 9a, we also report the query time of the exact algorithm `ER-I`. In general, all these approximation algorithms run much faster than `ER-I`, and their actual average approximation ratios and failure probabilities are all much lower than the theoretical values. One exception is that the failure probability of `ER-I-A`$_2$ on D5 (as-Skitter) is much larger than its theoretical value, because the value of ω in `ER-I-A`$_2$ is too small ($\omega = 2$) which leads to a high variance. Specifically, `ER-I-A`$_2$ is the fastest approximation algorithm on all datasets, but its actual approximation ratio and failure probability are the highest. Although `ER-I-A`$_5$ is the slowest, it achieves the best actual approximation ratio and failure probability. We observe that a larger ω leads to higher accuracy. We also see that `ER-I-A`$_3$ achieves the best balance among the running time, approximation ratio, and failure probability. We thus suggest to set ω to 3 (with $r = 5$ and $\delta = 0.01$).

7. CONCLUSIONS

In this paper, we examine the minimal SMCS problem. We develop Expand-Refine algorithms for finding minimal SMCSs. In addition, we propose two strategies to further improve the efficiency by relaxing connectivity and minimality. Our experiments on large datasets demonstrate the effectiveness and efficiency of our proposed solutions. We plan to extend our techniques to cohesive subgraph search under other metrics.

Acknowledgments

Reynold Cheng, Jiafeng Hu, Siqiang Luo and Yixiang Fang were supported by the Research Grants Council of Hong Kong (RGC Projects HKU 17229116 and 17205115) and the University of Hong Kong (Projects 102009508 and 104004129). Xiaowei Wu was supported by the Hong Kong RGC grant 17217716. We would like to thank the reviewers for their insightful comments.

8. REFERENCES

[1] T. Akiba, Y. Iwata, and Y. Yoshida. Linear-time enumeration of maximal k-edge-connected subgraphs in large networks by random contraction. In *CKIM'13*, pages 909–918, 2013.

[2] O. Amini, D. Peleg, S. Pérennes, I. Sau, and S. Saurabh. Degree-constrained subgraph problems: Hardness and approximation results. In *WAOA*, 2008.

[3] S. Asthana, O. D. King, F. D. Gibbons, and F. P. Roth. Predicting protein complex membership using probabilistic network reliability. *Genome Research.*, 14:1170–1175, 2004.

[4] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data Min. Knowl. Discov.*, 29(5):1406–1433, Sept. 2015.

[5] M. G. Bell and Y. Iida. *Transportation network analysis*. 1997.

[6] L. Chang, X. Lin, L. Qin, J. X. Yu, and W. Zhang. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *SIGMOD'15*, pages 459–474, 2015.

[7] L. Chang, J. Yu, and L. Qin. Fast maximal cliques enumeration in sparse graphs. *Algorithmica*, 66(1):173–186, 2013.

[8] L. Chang, J. X. Yu, L. Qin, X. Lin, C. Liu, and W. Liang. Efficiently computing k-edge connected components via graph decomposition. In *SIGMOD'13*, pages 205–216, 2013.

[9] J. Cheng, Y. Ke, S. Chu, and M. Ozsu. Efficient core decomposition in massive networks. In *ICDE*, 2011.

[10] J. Cheng, Y. Ke, A. W.-C. Fu, J. X. Yu, and L. Zhu. Finding maximal cliques in massive networks by h*-graph. In *SIGMOD'10*, pages 447–458, 2010.

[11] N. Chiba and T. Nishizeki. Arboricity and subgraph listing algorithms. *SIAM Journal on Computing*, 14(1):210–223, 1985.

[12] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *National Security Agency Technical Report*, 2008.

[13] C. J. Colbourn and C. Colbourn. *The combinatorics of network reliability*, volume 200. Oxford University Press New York, 1987.

[14] W. Cui, Y. Xiao, H. Wang, Y. Lu, and W. Wang. Online search of overlapping communities. In *SIGMOD*, pages 277–288, 2013.

[15] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local search of communities in large graphs. In *SIGMOD*, 2014.

[16] Y. Fang, R. Cheng, S. Luo, and J. Hu. Effective community search for large attributed graphs. *Proc. VLDB Endow.*, 9(12):1233–1244, 2016.

[17] A. Gibbons. *Algorithmic Graph Theory*. Cambridge University Press, 1985.

[18] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.

[19] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *Proc. VLDB Endow.*, 9(4):276–287, Dec. 2015.

[20] R.-H. Li, L. Qin, J. X. Yu, and R. Mao. Influential community search in large networks. *Proc. VLDB Endow.*, 8(5):509–520, Jan. 2015.

[21] R.-H. Li, J. Yu, and R. Mao. Efficient core maintenance in large dynamic graphs. *TKDE'14*, 26(10):2453–2465, Oct 2014.

[22] M. Maresca and H. Li. Connection autonomy in simd computers: a vlsi implementation. *Journal of Parallel and Distributed Computing*, 7(2):302–320, 1989.

[23] K. Mehlhorn. A faster approximation algorithm for the steiner problem in graphs. *Information Processing Letters*, 27(3):125–128, 1988.

[24] L. Qin, R.-H. Li, L. Chang, and C. Zhang. Locally densest subgraph discovery. In *KDD'15*, pages 965–974, 2015.

[25] S. B. Seidman. Network structure and minimum degree. *Social Networks*, 5(3):269 – 287, 1983.

[26] M. Sozio and A. Gionis. The community-search problem and how to plan a successful cocktail party. In *KDD*, 2010.

[27] V. Spirin and L. A. Mirny. Protein complexes and functional modules in molecular networks. *Proceedings of the National Academy of Sciences*, 100(21):12123–12128, 2003.

[28] J. Wang and J. Cheng. Truss decomposition in massive networks. *Proc. VLDB Endow.*, 5(9):812–823, 2012.

[29] S. Wasserman and K. Faust. *Social Network Analysis: Methods and Applications*. Cambridge University Press, 1994.

[30] D. R. White and F. Harary. The cohesiveness of blocks in social networks: Node connectivity and conditional density. *Sociological Methodology*, 31(1):305–359, 2001.

[31] Y. Wu, R. Jin, J. Li, and X. Zhang. Robust local community detection: On free rider effect and its elimination. *Proc. VLDB Endow.*, 8(7):798–809, 2015.

[32] Z. Zeng, J. Wang, L. Zhou, and G. Karypis. Out-of-core coherent closed quasi-clique mining from large dense graph databases. *ACM Trans. Database Syst.*, 32(2), 2007.