Size-constrained Community Search on Large Networks: An Effective and Efficient Solution

Fan Zhang, Haicheng Guo, Dian Ouyang, Shiyu Yang, Xuemin Lin, and Zhihong Tian

Abstract—As a fundamental graph problem, community search is applied in various areas, e.g., social networks, the world wide web, and biology. A common requirement from real applications is to return a community with a bounded size while most existing solutions do not constrain community size. Recent studies on size-constrained community search still have some critical issues, e.g., the existence of a better cohesiveness objective, some queries returning empty results, and inefficiency on partial queries. Thus, in this paper, we study the size-constrained truss community search (STCS). Given a graph G, a query vertex q, and size constraint $[l, h]$, the STCS problem aims to find a subgraph containing q with the largest min-support among all connected subgraphs having at least l and at most h vertices. We prove the STCS problem is NP-hard and APX-hard unless P=NP. An effective heuristic is proposed to quickly find a high-quality initial result. Then, a branch and bound algorithm is introduced to find the exact result, with novel optimizations, e.g., budget-cost-based bounding and branching strategies. Extensive experiments verify that the community quality returned by our algorithm is better and our algorithm is faster by up to 5 orders of magnitude, compared with the state-of-the-art.

Index Terms—Cohesive subgraph, community search, k-truss, size constraint

1 Introduction

Real-world networks are often modeled as graphs to process the complex relations among different entities, e.g., social networks, transaction networks, the world wide web, and biological networks. Cohesive subgraph is a fundamental concept in graph processing and is widely adopted in the study of community discovery [1]. As surveyed in [2], users are usually more interested in the communities they engage in, and community search is widely studied to find the communities containing specified query vertices. Community search studies enjoy a large spectrum of applications, e.g., team formation [3], event organization [4], social contagion modeling [5], social marketing [6], and protein function identification [7].

Although community search is extensively studied in the literature, the natural constraint on community size is not full-filled by most existing algorithms [2]. When there is a limit on budget or capacity, it is reasonable to restrict the size of the returned community to accommodate the application scenario [8], [9]. Thus, size-constrained community search is recently studied to find a cohesive subgraph containing query vertices and satisfying user-specified size constraints. The state-of-the-art studies are the SCS problem [10] built on k-core and the SCkT problem [11] built on k-truss. Nevertheless, existing works still suffer from several important issues, e.g, there exists a better cohesiveness objective, the hard cases are not well addressed, the parameter on cohesiveness is hard to be determined and some queries may return empty result.

Let *min-degree* of a subgraph S denote the smallest degree of a vertex in S. Given a graph, a query vertex q, and size constraint $[l, h]$, the SCS problem [10] is to find a subgraph with the largest min-degree among every connected subgraph containing q that has at least l and at most h vertices. The SCS problem is parameter-free on cohesiveness setting, and the result is not empty for any proper size constraint values. Nevertheless, a major issue of the SCS problem is the cohesiveness objective based on vertex degree can be improved. For community-related studies, it is more promising to adopt a triangle-based cohesiveness objective. For instance, the k-truss model built on *edge support* (the number of triangles containing the edge) is recognized as a strengthened version of the k-core model based on vertex degree [12]–[14]. The k-truss is better connected in structure because it is always a cohesive subgraph inside the $(k-1)$-core.

The triangles represent strong and stable vertex relations because the vertices inside have certain common neighbors [15]–[17]. Thus, it is more promising to adopt the "support" metric as the cohesiveness objective. Besides, it is hard to efficiently address the hard cases of the SCS problem. Given a time limit of 2 hours, the percentage that the SOTA algorithm (SC-BRB) [10] returns an exact result is about 69.3% for querying a random vertex with size constraint $[11, 20]$ on the 12 real graphs in our experiments.

Given a graph G, an integer k, a query vertex q and a size upper bound h, the SCkT problem [11] aims to find a triangle-connected k-truss subgraph containing q with size not exceeding h. The first issue of the SCkT problem is the selection of k. Although the input of different k can adjust the cohesiveness requirement, it is often difficult to select a proper k in many applications. Besides, the result of the SCkT problem may be empty for some queries due to the

- *F. Zhang, H. Guo, D. Ouyang, S. Yang and Z. Tian are with the Cyberspace Institute of Advanced Technology, Guangzhou University, China. E-mail: {zhangf, dian.ouyang, syyang, tianzhihong}@gzhu.edu.cn, haicheng_guo@126.com.*
- *X. Lin is with the Antai College of Economics and Management, Shanghai Jiao Tong University, China. E-mail: xuemin.lin@sjtu.edu.cn.*

Manuscript received; revised.

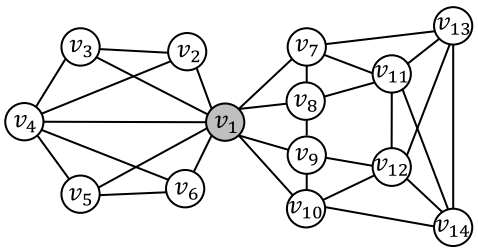


Fig. 1. Example with Query Vertex v_1 and Size Constraint $[6, 9]$

constraint by a fixed k. For instance, the vertex set of all the 5-trusses occupies the whole vertex set of G by 0.43 on average, for the 12 real graphs deployed in our experiments. Thus, a random query of SCkT may return empty for at least $1 - 0.43 = 57\%$ cases when $k = 5$.

Motivated by the above observations, the cohesiveness objective of our proposed size-constrained truss community search (STCS) problem is to maximize the min-support of the resulting subgraph. To ensure our model is vertex-oriented, the *min-support* is defined as the minimum vertex support, and the support of a vertex is the largest support of an edge incident to the vertex. Given a graph, a query vertex q, and size constraint $[l, h]$, *the STCS problem* is to find a subgraph with the largest min-support among every connected subgraph containing q that has at least l and at most h vertices. Our STCS problem is parameter-free on the cohesiveness constraint and would not return an empty result as long as there is a connected subgraph including q and with size in $[l, h]$. As a fundamental problem with general settings, the STCS model would benefit the applications of size-constrained community search, e.g., team formation, biological module identification, and the discovery of suspicious groups [11].

Example 1. *Given the graph in Figure 1 with query vertex v_1 and size constraint $[6, 9]$, our STCS problem will return the subgraph S_1 induced by $\{v_1, v_2, ..., v_6\}$ with min-support of 2 and size of 6, and the SCS problem will return the subgraph S_2 induced by $\{v_1, v_7, v_8, ..., v_{14}\}$ with min-degree of 4 and size of 9. Note that the min-degree of S_1 is 3 and the min-support of S_2 is 1. The result of STCS (S_1) is more promising because the vertices in S_1 are better connected and are closer to v_1, compared with the result of SCS (S_2).*

Challenges and Our Solution. To the best of our knowledge, this paper is the first work to study the problem of STCS, and the existing algorithms cannot solve our problem. The techniques for SCS problem [10] are based on the degree constraint which are different to the support constraint in our STCS problem. Besides, the support constraint k is an input of SCkT [11], while our STCS aims to optimize the min-support and the optimal min-support is various on different queries. We prove that the STCS problem is NP-hard and even APX-hard unless P=NP. As in recent studies, the main challenge of size-constrained community search is the existence of hard cases. Due to the problem hardness, it is unpromising to theoretically guarantee the efficient processing of all the hard cases. However, we observe that some hard cases are not essentially difficult because they derive from incorrect initial search branches.

In our branch and bound exact algorithm, we first design an effective heuristic to initialize a high-quality subgraph that is close to the optimal result. As the diameter of a k-truss is bounded by k and its size [18], we can compute the initial subgraph on the structure close to query vertex, i.e., an expansion based approach is preferred. The result of the heuristic can be returned within a short time limit and the approximate lower bound to the optimal min-support is derived. Then, we propose novel bounding techniques on vertex budget-cost relations of current partial solution and candidate set to early terminate unpromising branches. Accordingly, the reduction and branching techniques are designed to judiciously determine the search branch and narrow down the search space.

Contributions. The principal contributions of the paper are summarized as follows.

- To our best knowledge, it is the first work to study the size-constrained truss community search (STCS) problem. We analyze the defects in existing studies in-depth and motivate the STCS problem. We prove that the STCS problem is NP-hard and hard to approximate.
- An effective heuristic algorithm is proposed to initialize a high-quality subgraph and thus speed up the exact search. A branch and bound algorithm is proposed to find the optimal result, with a novel budget-cost based bounding for pruning and well-designed techniques on reduction and branching.
- Extensive experiments on 12 real-life graphs verify that the quality score of the community returned by STCS is better, many (fake) hard cases are addressed, the time-dependent results of STCS algorithm are of high-quality, and the runtime of our exact search is faster by up to 5 orders of magnitude, compared with the state-of-the-art solutions.

2 Related Work

Cohesive Subgraph Models. With different requirements from application scenarios, various cohesive subgraph models are studied, including clique [19], [20], quasi-clique [21], [22], k-core [23]–[27], k-truss [12], [13], [18], [28], and k-ecc [29], [30]. The challenges of large graph processing are summarized in [31]. Cohesive subgraph models are surveyed in [1], [32], [33], and the models can be extended to handle other graph types, such as heterogeneous information networks [34], [35] and geo-social networks [36]. The k-truss is always a subgraph of $(k-1)$-core or $(k-1)$-ECC but not vice versa [18]. Besides, a k-truss with n vertices has a diameter within $\frac{2n-2}{k}$.

Due to the elegant cohesiveness constraint and diameter-bounded property, k-truss (or min-support constraint) are widely adopted for the community search problem, e.g., [13], [14], [37]–[39]. Recently, k-truss based community search is also studied on bipartite graphs [40], [41] and directed graphs [42]. Nevertheless, the above models are not designed to solve the size-constraint community search.

Size-constrained Community Search. The studies of community search on different graphs are surveyed in [2]. Derives from the natural limit of budget or capability, the search of size-constrained (i.e. size-bounded) community

TABLE 1
Summary of Notations (We may omit G when the context is clear, e.g., $N_G(v)$ may be abbreviated to $N(v)$.)

Notation	Definition
$N_S(u)$	the neighbor set of vertex u in S
$deg_S(u)$	the degree of vertex u in S
q	the query vertex
$l; h$	lower/upper bound of subgraph size
L	a clique containing q
$N_S(V)$	the union of $N_S(u)$ for every vertex $u \in V$
$sup_S(u,v)$	the edge support of (u,v) in S, i.e., $\lvert\{w \mid w \in V(S) \wedge \triangle(u,v,w) \in S\}\rvert$
$sup_S(u)$	the vertex support of u in S, i.e., $\max\{sup_S(u,v) \mid v \in N_S(u)\}$
C	a partial solution (vertex set)
R	a candidate vertex set
$\tau_S(u,v)$	the trussness of edge (u,v) in S
$\tau_S(u)$	the trussness of vertex u in S, i.e., $\max\{\tau_C(u,v) \mid v \in N(u) \cap V(S)\}$
$T_S^q(k)$	the connected subgraph of S containing q with minimum vertex trussness k
$N_S^{\geq k}(u)$	the set $\{v \mid v \in N_S(u) \wedge sup_G(v) \geq (k-2)\}$
$N_S^{\geq k}(C)$	the union of $N'^{\geq k}_S(u)$ for every vertex $u \in C$
$k'; k^*$	lower/upper bound of "the optimal min-trussness (minimum vertex trussness of an optimal result)"

is studied recently. A heuristic algorithm is proposed in [8] to recursively remove the vertices from a large initial subgraph. A local search algorithm is designed to expand a subgraph from the query vertex in a greedy way [43]. In [44], the local search algorithm computes a k-core with size equal to h and the smallest closeness among all size-h k-core subgraphs. In [43] and [45], the search of minimum communities is studied to further shrink the size of returned communities. A solution for finding a minimum k-core is developed with a size approximation guarantee [46]. The state-of-the-art solutions on size-constrained community search are SCkT [11] and SCS [10]. As discussed in the introduction, they still suffer from certain defects and our STCS problem aims to address these defects with a practically-efficient exact algorithm.

It is infeasible to apply the above algorithms to our problem, because the above models are all based on the k-core model except SCkT. One reason is that our STCS does not need a k-core as a seed subgraph for computing k-truss. We can simply compute the truss decomposition [12] as pre-processing for immediate retrieval of a k-truss. The other reason is that the initial step of SCkT algorithm has a different aim with our proposed algorithm. SCkT first computes a size-constrained subgraph on a k-truss with given k value [11], while our STCS problem should initialize a subgraph with a large min-support value corresponding to optimizing the k value. Besides, as the objective of our STCS problem is different from other problems, the effective branching strategy and bounding techniques need a sophisticated design according to our new objective.

3 PRELIMINARIES

In this section, we introduce the notations, formally define the problem, and prove the problem hardness.

3.1 Notations and Definitions

We consider an undirected simple graph $G = (V, E)$, with $n = |V|$ vertices and $m = |E|$ edges (assume $m > n$). The set of vertices (resp. edges) of graph G is denoted by $V(G)$ (resp. $E(G)$). Let S be a subgraph of G induced by vertex set M, i.e., we have $V(S) = M$ and $E(S) = (M \times M) \cap E(G)$. For a vertex $v \in V(S)$, the set of v's neighbors in S is denoted by $N_S(v) = \{u \mid (u,v) \in E(S)\}$. The degree of v in S is denoted by $deg_S(v) = |N_S(v)|$. A triangle $\triangle_{u,v,w}$ is a cycle of three in G such that $\{(u,v),(v,w),(w,u)\} \in E(G)$. Table 1 summarizes the notations in the paper.

The k-truss model is defined on edge support [18].

Definition 1. ***Edge Support.*** *The support of an edge (u,v) in S, denoted by $sup_S(u,v)$, is the number of the triangles in S with each containing (u,v), that is, $sup_S(u,v) = |\{w \mid w \in V(S) \wedge \triangle(u,v,w) \in S\}|$.*

Definition 2. ***k-Truss.*** *Given a graph G and an integer k, a subgraph S is a k-truss of G, if (i) each edge $(u,v) \in E(S)$ is contained in at least $k-2$ triangles in S, i.e., $sup_S(u,v) \geq k-2$; (ii) S is connected; and (iii) S is maximal, i.e., any supergraph of S is not a k-truss except S itself.*

The minimum support constraint in k-truss ensures that every edge in the k-truss is a strong tie. Thus, the k-truss subgraphs are dense and stable on structure, leading to various studies, e.g., [2], [11], [13], [14], [37]–[39], [47].

Because real-world communities are usually vertex-oriented [48], i.e., the existence of edges is determined by their incident vertices, a more promising concept for community discovery is the support value on each vertex.

Definition 3. ***Vertex Support.*** *The support of a vertex $u \in V(S)$, denoted by $sup_S(u)$, is the largest support of its incident edge, that is, $sup_S(u) = \max\{sup_S(u,v) \mid v \in N_S(u)\}$.*

In most real communities, the weak ties between community members are kept as a natural part of the communities [49], while the k-truss removes all the weak ties inside (the edges with support less than $k-2$). It is also validated that computing a vertex-oriented subgraph is more efficient and effective than edge-oriented subgraph like k-truss [48]. Consequently, in this paper, we adopt vertex support as the cohesiveness objective of our problem. Let **min-support** of S represent the minimum vertex support in S.

Problem Statement. Given a graph $G = (V, E)$, a query vertex $q \in V$, and a size constraint $[l, h]$, the Size-constrained Truss Community Search (STCS) problem aims to find a subgraph H of G such that it satisfies all the following conditions:

(1) Connectivity: H is connected and contains q;

(2) Size Constraint: the size of H is constrained, i.e., $l \leq |V(H)| \leq h$;

(3) Cohesiveness: The min-support of H is maximized, i.e., $\min\{sup_H(u) \mid u \in V(H)\}$ is maximized, among all subgraphs of G satisfying the above two conditions.

An example of the STCS problem is shown in Example 1.

3.2 Problem Hardness

We investigate the hardness of the STCS problem.

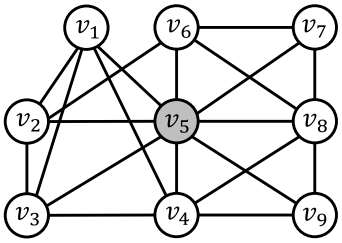


Fig. 2. Running Example of STCS

Theorem 1. *The STCS problem is NP-hard.*

Proof. We prove this by reducing the k-clique problem to the decision version of STCS problem. Given a graph G and an integer k, the k-clique problem is to check whether G contains a clique of size k, which is NP-complete [50]. The decision STCS problem is as follows: given a graph G, a query vertex $q \in V(G)$, a size constraint $[l, h]$, and an integer k, determine whether G has a subgraph H such that (1) H is connected and contains q, (2) $l \leq |V(H)| \leq h$, and (3) the minimum vertex support of H is at least $k-2$.

For an instance of the k-clique problem, we reduce it to a decision STCS problem with $q = \emptyset$, size constraint $[k, k]$ and support threshold $k-2$. Clearly, any clique with k vertices is a connected subgraph H with size k and minimum vertex support $k-2$, i.e., the subgraph asked by decision STCS problem. On the other hand, any solution to the decision STCS problem is a clique of size k. Thus, the decision STCS is NP-complete and the optimization version of STCS problem is NP-hard. □

Theorem 2. *The STCS problem is APX-hard, unless P=NP.*

Proof. We consider a graph G, a query vertex q and a size constraint $[l, h]$. Assume our STCS problem is not APX-hard, i.e., it admits a PTAS. Thus, for any $\epsilon > 0$, there is a PTIME solution for STCS problem on G with size constraint $[l, h]$ and min-support $s' \geq (1-\epsilon) \times s^{opt}$ where s^{opt} is the optimal min-support. The above solution with each ϵ value solves the decision STCS problem which is NP-complete, proved in the proof of Theorem 1, i.e., we can determine in PTIME whether G has a qualified subgraph H with size constraint $[0, h]$ and $k = s' + 2$. Note that the value of ϵ corresponds to any possible input of k, leading to a contradiction with the NP-completeness. Thus, the STCS problem does not admit a PTAS and it is APX-hard unless P=NP. □

4 Our Solution

As it is the first work to investigate the STCS problem, we aim to propose a practically-efficient exact algorithm. The hardness of the STCS problem implies that there are some hard cases deriving from the size constraint and the optimization of the min-support. As discussed in Section 2, existing studies on size-constrained community search cannot solve our problem due to the different objectives. In the following, we will introduce an effective heuristic ST-Heu in Section 4.1 and then propose the exact search algorithm ST-Exa in Section 4.2.

We first introduce the concept of vertex/edge trussness used in our algorithms.

Definition 4. ***Edge Trussness.*** *The trussness of an edge (u, v) in $E(G)$, denoted by $\tau(u, v)$, is the largest k such that a k-truss of G contains (u, v).*

Given the trussness of each edge, the k-truss is formed by all the edges with trussness no less than k and their endpoints.

Definition 5. ***Vertex Trussness.*** *The trussness of a vertex $u \in V(G)$, denoted by $\tau(u)$, is the largest trussness among $\{\tau(u, v) \mid (u, v) \in E(G)\}$.*

For an integer k, the vertex set of all the k-trusses is equal to the set of vertices with trussness no less than k. Let **min-trussness** of S represent the minimum vertex trussness in S. Given a subgraph S, the min-trussness of S is the same as the min-support of S plus 2. Thus, the objective function of the STCS problem can also be min-trussness of H.

Example 2. *For the graph in Figure 2, the trussness of edge (v_2, v_6) is 3 and the trussness of any other edge is 4. The trussness of each vertex is 4 as the largest trussness of its incident edge is 4.*

4.1 The Heuristic

We observe an effective heuristic is essential for the efficiency of the exact search as the initialization. A well-designed heuristic can fast produce a high-quality result, and thus avoid the visit of unpromising search branches. Different from the design of existing heuristics [8], [10], [43], [45], [46], we first compute the upper bound of optimal min-trussness k^* of the exact result s.t. the initial result computed according to k^* can be of high quality. During the computation of the initial result, we design a scoring function to guide the selection of each vertex, which considers both local connections to the vertex and its potential in participating in a dense subgraph in a global view. We further optimize the basic heuristic by alleviating the issues of "slow start" (scoring function may degrade at the beginning stage) and "branch trap" (the search may fall into a wrong branch due to the undiversified initial subgraph). In the overall design, we ensure that the returned subgraph is not empty as long as there is a connected subgraph in G that includes the query vertex and satisfies the size constraint.

In the heuristic, we first conduct a truss decomposition of G by one time, as a preprocessing step to compute the trussness of each edge. Truss decomposition is to recursively remove each edge with the smallest support, which costs $O(m^{1.5})$ time [12].

Basic Heuristic (ST-Base) Algorithm 1 shows the pseudo-code of our basic heuristic algorithm. Let **optimal min-trussness** represent the min-trussness of the optimal result H, i.e., the min-support of H plus 2. Firstly, we compute an upper bound of optimal min-trussness, denoted by k^*, such that a seed subgraph can be generated according to k^*. Let $T^q(k)$ denote the connected subgraph of G containing q with min-trussness k, computed by traversing the vertices with trussness no less than k from q.

If the vertex trussness of q is larger than h (Lines 1-2), we set k^* to h because the size of k-truss is at least k and a subgraph of k^*-truss may violate the size constraint with a larger k^*. Here we initialize C by q; Otherwise (Lines 3-6), we set k^* to the largest k such that the size of $T^q(k)$ is

Algorithm 1: ST-Base

Input : a graph G, the trussness of each edge, size constraint [l,h], query vertex q
Output : the subgraph H containing q with size constraint $[l, h]$ and min-trussness k', the upper bound of optimal min-trussness k^*

1 **if** $\tau(q) > h$ **then**
2 $\quad k^* \leftarrow h$; $C \leftarrow \{q\}$;
3 **else**
4 $\quad k^* \leftarrow k$ s.t. $|V(T^q(k))| \geq l$ and $|V(T^q(k+1))| < l$;
5 $\quad$ **if** $|V(T^q(k^*))| \leq h$ **then return** $T^q(k^*)$;
6 $\quad C \leftarrow V(T^q(k^*+1)) \cup \{q\}$;
/* Add vertices into C and compute the k'-truss of C */
7 $R \leftarrow N_G^{\geq k^*}(C) \setminus C$;
8 **while** $|C| < h$ **do**
9 $\quad v \leftarrow$ **Score**(C, R);
10 $\quad C \leftarrow C \cup \{v\}$; $R \leftarrow R \setminus \{v\}$;
11 $\quad R \leftarrow R \cup N_G^{\geq k^*}(v) \setminus C$;
12 $H \leftarrow T^q_{G[C]}(k')$ s.t. $k' \leftarrow \max\{k \mid |V(T^q_{G[C]}(k))| \geq l\}$;
13 **return** H, k^*

not less than l (Line 4). When the size of $T^q(k)$ is no larger than h, we simply return $T^q(k)$ as an optimal result (Line 5); Otherwise, we initialize C by the vertex set of $T^q(k^*+1)$ and query vertex q as the seed subgraph (Line 6).

Let $N_S^{\geq k}(u)$ denote the set of u's neighbors in $S \subset G$ with vertex support not less than $k-2$ in G, i.e., $\{v \mid v \in N_S(u) \wedge sup_G(v) \geq (k-2)\}$. Let $N_S^{\geq k}(C)$ denote the union of $N_S^{\geq k}(u)$ for every vertex $u \in C$. The candidate vertex set R is initialized by $N_G^{\geq k^*}(C)$ minus the set C (Line 7). Next, we recursively move a promising vertex from R to C according to a scoring function, until the size of C is equal to h (Lines 8-11). Finally, we return a k'-truss of the subgraph induced by C with the largest k' such that the size of k'-truss is not less than l (Line 12).

The returned H is not empty as long as there exists a connected subgraph that includes q and satisfies the size constraint (i.e., the optimal result of STCS is not empty). If we have $k' = k^*$, the returned H is an optimal result of STCS problem, since k^* is an upper bound of the optimal min-trussness.

Note that, in this paper, some weak ties (support less than $k'-2$) between the vertices in H may not exist in H. We may add these weak ties to H and the updated H is also a qualified approximate or exact result.

Scoring Function. Selecting a promising vertex from R in each iteration (Line 9) is critical to find a high-quality initial subgraph H. Intuitively, each selected vertex should satisfy two conditions: (1) from the local view, it needs to connect more vertices in C; and (2) from the global view, it exists in a subgraph of G with higher cohesiveness. Thus, our scoring function in the heuristic is as follows.

Definition 6. *Given a seed set C and a candidate set R of Algorithm 1, **Score**(C, R) returns a vertex v with the highest score computed as follows.*

$$Score(C, R) = \arg\max_{v \in R}\{v \mid |N^{\geq k^*}_{G[C \cup \{v\}]}(v)| + \tfrac{\tau_G(v)}{k_{max}+1}\}$$

where k_{max} is the largest vertex trussness in G.

In the scoring function, we find the vertex v from R with the most neighbors in C and their trussness in G is no less than k^*, i.e., with the highest $|N^{\geq k^*}_{G[C\cup\{v\}]}(v)|$, and ties are broken by the trussness of each vertex, i.e., by $\tau(v)/(k_{max}+1)$. Note that the above degree based score is adopted other than a support based score for two reasons: (i) a support based score is more costly due to the triangle counting; and (ii) the seed subgraph C may not contain many triangles.

We observe that the above basic algorithm may suffer from the issue of **Slow Start**: when $C = \{q\}$ at Line 2 or Line 6 of Algorithm 1 (i.e., $T^q(k^*+1)$ is overlarge or empty), the score of every vertex $v \in R$ may be small at this beginning stage, and thus the effectiveness of the scoring function may be degraded.

ST-Cli. To solve the slow start issue, we propose the ST-Cli algorithm. The only difference between ST-Cli and ST-Base is the initialization of C. In ST-Cli, if $C = \{q\}$ after Line 6 of Algorithm 1, we first initialize C by a clique containing q and then run Lines 7-13. The reasons for using a clique are as follows: (i) the clique containing q certainly exists, which is a basic unit for every query vertex; (ii) the clique will not be too small as long as q has some close neighbors; (iii) a relatively large C initialized by clique can enhance the scoring function, because the score of each vertex (built on the neighbors in C) can be better distinguished from each other; and (iv) it is costly to find the maximum clique or enumerate all the maximal cliques.

To fast compute a qualified clique L, we initialize L by q and append L with q's 1-hop neighbors. Specifically, we add each vertex $v_i \in \{N_G^{\geq k^*}(q) \setminus L\}$ satisfying $L \subseteq N(v_i)$, i.e., a neighbor of q with trussness no less than k^*, not in L, and adjacent to every vertex of L. Such a vertex is called a qualified vertex. Each qualified vertex v_i is added to L according to a non-increasing order of $|N(v_i) \cap N_G^{\geq k^*}(q)|$, because such a v_i is likely to form a large clique. The appending process is terminated when no qualified vertex can be added to L or the size of L reaches h.

We further observe that initializing the branch and bound algorithm based on a single clique may face the issue of **Branch Trap**: the search may fall into a wrong branch following the initial single clique. To alleviate this issue, we compute diversified cliques to achieve a better initial result.

Advanced Heuristic (ST-Heu). The pseudo-code of the advanced heuristic ST-Heu is given in Algorithm 2. We firstly execute Lines 1-6 of Algorithm 1 to find the upper bound k^* of the optimal min-trussness. When $C \neq \{q\}$ (Line 2), the subgraph $T^q(k^*+1)$ is certainly not empty or overlarge, and thus we continue the execution of Algorithm 1 at Line 3 where $T^q(k^*+1)$ is the seed subgraph to be expended. When $C = \{q\}$ (Lines 4-15), we address the "slow start" and "branch trap" issues based on diversified cliques.

We use D to record the set of vertices which are already covered by a clique (Lines 5 and 9). At Line 6, we visit each neighbor v_i of query vertex q with trussness no less than k^* in G (i.e., the promising candidate to join C), by non-increasing order of the common neighbor number of v_i and q (with trussness no less than k^*). The front vertices in the above order are likely to form a dense initial subgraph together with q. Then, we compute the diversified cliques on the subgraph S induced by the set of above common

Algorithm 2: ST-Heu

Input : a graph G, the trussness of each edge, size constraint [l,h], query vertex q
Output : the subgraph H containing q with size constraint $[l, h]$ and min-trussness k', the upper bound of optimal min-trussness k^*

1 Lines 1-6 of Algorithm 1;
2 **if** $C \neq \{q\}$ **then**
3 Lines 7-13 of Algorithm 1;
4 **else**
  /* Address "slow start" and "branch trap" */
5 $D \leftarrow \emptyset$;
6 **for** each $v_i \in N^{\geq k^*}(q) \setminus D$ in non-increasing order of $|N(v_i) \cap N^{\geq k^*}(q)|$ **do**
7 $S \leftarrow$ the subgraph induced by $N(v_i) \cap N^{\geq k^*}(q)$;
8 $\mathcal{L} \leftarrow$ the set of diversified cliques in S;
9 $D \leftarrow D \cup \mathcal{L}$; $k' \leftarrow 0$;
10 **for** each clique $L \in \mathcal{L}$ **do**
11 $C \leftarrow \{q, v_i\} \cup L$;
12 $H_i \leftarrow$ Lines 7-13 of Algorithm 1;
13 $k'_i \leftarrow$ min-trussness of H_i;
14 **if** $k'_i = k^*$ **then return** H_i, k^*;
15 **if** $k'_i > k'$ **then** $k' \leftarrow k'_i$; $H \leftarrow H_i$;
16 **return** H, k^*

Algorithm 3: ST-B&B

Input : a graph G, the trussness of each edge, size constraint [l,h], query vertex q, the upper bound of optimal min-trussness k^*, current largest min-trussness k' among each computed result, partial solution C, candidate vertex set R
Output : the optimal result H on the branch of C and R

1 **if** $k' = k^*$ **then return** H;
2 **if** $|C| \geq l$ **then**
3 $\hat{k} \leftarrow \max\{k \mid |V(T^q_{G[C]}(k))| \geq l\}$;
4 **if** $\hat{k} > k'$ **then** $k' \leftarrow \hat{k}$; $H \leftarrow T^q_{G[C]}(\hat{k})$;
5 **if** $|C| < h$ and $|R| \neq 0$ **then**
6 $v^* \leftarrow$ the vertex v in R with the highest score on $|N^{\geq k'+1}_{G[C\cup\{v\}]}(v)| + \frac{\tau(v)}{k_{max}+1}$;
7 **ST-B&B**(..., $C \cup \{v^*\}$, $R \setminus \{v^*\}$);
8 **ST-B&B**(..., C, $R \setminus \{v^*\}$);
9 **return** H

neighbors (Lines 7-8). As the size of S is not large, the cost of clique search is minor in our heuristic algorithm. At Line 8, we can adopt the state-of-the-art algorithm to find the set of diversified top-x maximal cliques (with size limit of h), denoted by $\mathcal{L}$ [51]. In our experiments, the resulting min-trussness of H when $x = 2$ is very close to (same on about 99.4% cases) the min-trussness by enumerating all the maximal cliques.

For each clique L of $\mathcal{L}$, we set the seed subgraph C by the union of L and $\{q, v_i\}$ (Lines 10-11), and execute Lines 7-13 of Algorithm 1 to expand C with the promising vertices in R (Line 12). Once the min-trussness of current result H_i is equal to the upper bound k^*, H_i is an optimal result (Lines 13-14). We will update H and its min-trussness k' if a better result is found (Line 15). The algorithm returns when each candidate vertex is covered by the cliques and no more clique can be expanded as the seed subgraph.

Example 3. *Given graph G in Figure 2, query vertex v_5 and size constraint $[8, 8]$, by ST-Base, we obtain that the upper bound of min-trussness $k^* = 4$ (Line 4 of Algorithm 1) and $R = V(G) \setminus \{v_5\}$ (Line 7). ST-Base may first select v_1 as the score of every vertex is $(1 + 0.8)$ by Definition 6 where $|N^{\geq k^*}_{G[C\cup\{v_1\}]}(v_1)| = 1$, $\tau(v_1) = 4$ and $k_{max} = 4$. Then, v_2 may be selected as it has the largest score $(2 + 0.8)$, followed by v_3, v_4, v_8, v_9 and v_6 with scores 3.8, 3.8, 2.8, 3.8 and 2.8, respectively. Finally, ST-Base returns the subgraph S' containing $V(G) \setminus \{v_7\}$ with min-trussness 3, in a slow start way.*

Correspondingly, ST-Cli will certainly compute on the clique L_1 consisting of $\{v_4, v_5, v_8, v_9\}$, where ST-Cli first selects $v_i = v_4$ or v_8 because $|N(v_i) \cap N^{\geq k^*}_G(v_5)| = 4$ is larger than that of any other vertex in R. Then, if v_1 is selected, the expansion from ST-Cli also returns S' with min-trussness 3, which may incur the branch trap in an exact search.

Our advanced ST-Heu further computes on clique L_2 consisting of $\{v_5, v_6, v_7, v_8\}$, and returns the subgraph S^* containing $V(G) \setminus \{v_2\}$ with the optimal min-trussness 4, where v_2 is not selected because $sup(v_2, v_6) = 1$ and the score of v_2 is smaller than other vertices. ST-Heu alleviates "slow start" and "branch trap" issues in this example.

Complexity. ST-Heu firstly computes k^* with time complexity of $O(m)$ because it iteratively visits the connected subgraphs including query vertex. The time complextiy of finding the diversified cliques is $O(d \cdot deg(q) \cdot 3^{d/3})$ where $d = \max_{v\in N(q)}(|N(q) \cap N(v)|)$ [51] and the computation is limited to 1-hop neighborhood of query vertex. Note that our implementation adopts top-2 diversified cliques for performance trade-off. The time complexity of trussness computation to find H is $O((\frac{hd_{max}}{2})^{1.5})$ where d_{max} is the maximum degree in G. Thus, the time complexity of ST-Heu is $O(d \cdot deg(q) \cdot 3^{d/3} + (\frac{hd_{max}}{2})^{1.5})$.

4.2 The Exact Search

Our advanced heuristic algorithm is likely to find a high-quality initial result that is often close to or equal to the optimal result. However, there are still some (real) hard cases derived from the hardness of the STCS problem. In the design of the exact solution, it is critical to propose novel bounding and branching techniques. Particularly, we consider the true connections between the candidate vertex set and the partial solution in each search step s.t. the bounding can be tight. For the branching strategies, we follow the scoring function in the heuristic to capture both local and global information, and introduce an effective backtracking technique. We also propose the reduction rules to further prune the search space.

In this section, we design an exact algorithm for the case of $k' \neq k^*$ when Algorithm 2 returns, i.e., the case we cannot guarantee the output H is an optimal result. We first introduce a basic branch and bound algorithm and then propose the novel bounding, reduction and branching techniques to improve the exact algorithm.

ST-B&B. The pseudo-code of ST-B&B is shown in Algorithm 3 which recursively searches all the branches with the partial set C and candidate set R. Note that C is initialized by only

the query vertex. ST-B&B utilizes the k' returned by Algorithm 1 as the largest min-trussness of current computed results to prune unpromising branches.

Each vertex in R may be moved into C or excluded from both C and R in the search branches. The input k^* is the upper bound of optimal min-trussness. The input k' is current best min-trussness, i.e., the largest one among the min-trussness of each computed subgraph H, where every H satisfies the connected and size constraint conditions of STCS while the optimality of cohesiveness (min-support) is not guaranteed. Note that the partial solution set C always contains query q and is connected as we expand C by adjacent vertices.

As we execute the heuristic algorithm before the exact search, we have $k^* \neq k'$ at the initial branch if the exact search is invoked. Once we have $k' = k^*$ (Line 1), the recursive algorithm is returned because the subgraph H is already an optimal result. When $|C| \geq l$, we use $\hat{k}$ to record the largest k such that the size of $\hat{k}$-truss of $G[C]$ containing q is not less than l (Lines 2-3). Then, we update the records accordingly if $\hat{k}$ is larger than k' (Line 4). Next, we recursively add a promising vertex to C from R until the size of C reaches h or R becomes empty (Line 5). Here, we adapt the metric in the scoring function (Section 4.1) to find a good v^* (Line 6). To enumerate all feasible subgraphs for query vertex q and the size constraint, the algorithm invokes the branches of $(C \cup \{v^*\}, R/\{v^*\})$ and $(C, R/\{v^*\})$ for the vertex $v^* \in R$ with the highest score (Lines 6-8).

However, ST-B&B is still inefficient to process the real hard cases, because no bound to prune unpromising branches, the candidate set can be very large, and the search order of branches is not well-designed. Thus, we introduce Budget-Cost Bounding, Reduction Rules and Branching Strategies in the following to optimize ST-B&B and propose the advanced ST-Exa algorithm.

SOTA Bounding Algorithms. Bounding techniques are critical in pruning unpromising branches for size-constrained community search. We consider the partial solution C and the candidate set R. Yao et al. [10] propose to compute $\hat{d}$: the upper bound of minimum degree in any subgraph computed on the branch of C and R satisfying the size constraint. Let d' denote the largest among the minimum degree of every qualified subgraph computed so far. If we have $\hat{d} \leq d'$, current search branch (C and R) is unpromising and can be terminated safely. Let b_{max} denote the maximum number of vertices we can move from R to C, i.e., $b_{max} = \min(h - |C|, |R|)$. The main idea to compute $\hat{d}$ is to link b_{max} vertices in R with the vertices in C based on solely their vertex degrees while the true connections (edges) between the vertices are ignored and the connections are reconstructed. To find a smaller $\hat{d}$, the paper decreases the number of edges to be reconstructed by removing the edges between large-degree vertices.

Liu et al. [11] propose a different bound for pruning unpromising branches, i.e., $\check{s}$: the size lower bound of any subgraph computed on C and R without violating the fixed trussness constraint k. If we have $\check{s} > h$, current search branch is unpromising and can be early-terminated. The main idea to compute $\check{s}$ is to find the largest number of vertices required in R for linking with every vertex in C based on its support gap to k. The edges between C and R are also reconstructed regardless of the true connections.

Our Budget-Cost based Bounding. We design a novel branch check algorithm (BranchCheck) which well exploits the structure information of C and R by computing the $cost$ and $budget$ of every vertex in C regarding the true connections with R. Let a **qualified subgraph** represent a subgraph satisfying the constraints of STCS except cohesiveness optimality. As we already find a qualified subgraph with min-trussness k' by ST-Heu or previous branches, the target for future search branches is to check whether we can find a qualified subgraph with min-trussness of at least $k' + 1$. Thus, for each vertex u in C, we consider whether the trussness of u can be $k' + 1$ in a result in the branch of C and R, where k' is current largest min-trussness among each computed result. Let $cost(u)$ represent the least number of vertices from R to be added into C such that the trussness of u can be $k' + 1$. Correspondingly, $budget(u)$ represents the maximum number of vertices we can move from R to C based on the true structure induced by C and R. If there is a vertex u with $budget(u) < cost(u)$, the branch is unpromising and can be terminated safely.

In the branch and bound search, once the partial solution C is updated, we execute Algorithm 4 to check whether the branch is promising. At Line 1, we first execute the SOTA truss maintenance [52] to update the trussness of each vertex u in C, i.e., update $\tau_{G[C]}(u)$ by $\max\{\tau_{G[C]}(u, v) \mid v \in N_{G[C]}(u)\}$. Then, we initialize the budget and cost of each vertex u in C. At Line 3, the budget of u is initialized to $min\{h - |C|, |N_G(u) \cap R|\}$. At Line 4, The visit tag of u is set to False. At Line 5, the cost of u is assigned by $\max\{k' + 1 - \tau_{G[C]}(u), 0\}$. If $budget(u) < cost(u)$, the branch is unpromising and thus terminated at Line 6. We use b_{min} and c_{max} to record the minimum budget and the maximum cost of a vertex in C, respectively (Line 7). As proved by Lemma 1, if $b_{min} \geq 2 \times c_{max}$, current branch is still promising and the branch continues at Line 8.

At Lines 9-15, we visit each vertex u in C and assume the trussness of u becomes $k' + 1$ if moving some vertices from R to C, where we update the budget of each affected vertex accordingly at Lines 13-15. For each vertex x in $C \setminus N(A)$ where $A = N(u) \cap R$, its budget may be reduced because u requires $cost(u)$ vertices moving from R to C and there is no common neighbor of u and x in R (Lines 15-16). That is, the trussness of u becoming $k' + 1$ will move some vertices from R to C while the vertex x has no benefit in trussness from these vertices, leading to a possible decrease of $budget(x)$. Thus, at Line 14, the budget of x is updated to $min\{h - |C| - cost(u), budget(x)\}$. Note that the update of $budget(x)$ is only based on one vertex in C for performance trade-off, i.e., we update by $h - |C| - cost(u)$ if it is smaller than $budget(x)$. That is, it cannot be paid-off if we use the cost of multiple vertices to reduce $budget(x)$. The search branch is terminated if the updated $budget(u)$ is less than $cost(u)$ (Line 15). We observe that the visiting order in C has a minor effect on the pruning power and adopt the order same to the vertex addition order in C.

Lemma 1. *If $b_{min} \geq 2 \times c_{max}$ holds in Line 8 of Algorithm 4, we have $budget(u) \geq cost(u)$ for each $u \in C$ suppose Lines 9-15 are executed, i.e., current search branch is promising.*

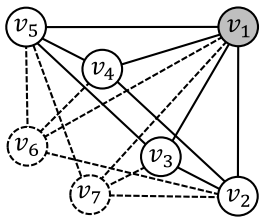


Fig. 3. Example of Budget-Cost based Bounding

Proof. The maximum cost of a vertex in C is always c_{max}. Before executing Lines 9-15, the minimum budget of a vertex in C is b_{min}. Suppose Lines 9-15 are executed, the budget of a vertex in C is at least $b_{min} - c_{max}$. If $b_{min} - c_{max} \geq c_{max}$, the budget of every vertex in C is not less than its cost. Thus, the current search branch is promising when $b_{min} \geq 2 \times c_{max}$. □

Example 4. *Given the graph G in Figure 3, a query v_1 and size constraint $[5, 6]$, suppose the partial solution set $C = \{v_1, v_2, v_3, v_4, v_5\}$, the candidate set $R = \{v_6, v_7\}$ and $k' = 3$. We examine whether C can be contained in a qualified subgraph with min-trussness of $k' + 1$. By truss maintenance, $\tau_{G[C]}(u)$ of each vertex $u \in C$ is 1. The budget and cost values of each vertex $u \in C$ are both 1, as $\min\{h - |C|, |N_G(u) \cap R|\} = 1$ and $\max\{k' + 1 - \tau_{G[C]}(u), 0\} = 1$. If v_3 is visited in Lines 9-15 of Algorithm 4, $budget(v_4)$ becomes 0 at Line 14 because $h - |C| - cost(v_3) = 1 - 1 = 0$. Thus, the current search branch is unpromising as $budget(v_4) = 0 < cost(v_4) = 1$.*

Complexity. By [52], the time complexity of truss maintenance in Line 1 of Algorithm 4 is $O((\frac{hd_{max}}{2})^{1.5})$, where d_{max} is the maximum degree in G. There are at most h iterations in Lines 9-15 for checking every vertex in C. The dominating time cost of checking one vertex is updating $budget(x)$ in Lines 11-15. It costs $O(d_{max})$ to find the neighbor set of u in R, i.e., $A \leftarrow \{N_G(u) \cap R\}$, and then $O(hd_{max})$ time to obtain $C \setminus A$. Thus, the time complexity of Algorithm 4 is $O((\frac{hd_{max}}{2})^{1.5} + h^2 d_{max})$.

Candidate Reduction. In order to reduce the size of candidate set, we try to delete the unpromising vertices in R, and add the promising vertices group from R to C without violating the size constraint.

To form a qualified subgraph with min-trussness of at least $k' + 1$, if the join of a vertex v from R to C requires adding more than $h - |C| - 1$ vertices to C, the vertex v is unpromising because the size constraint will be violated. As a vertex with trussness $k' + 1$ has at least k' neighbors in a $(k' + 1)$-truss, the following reduction rule holds.

Reduction Rule 1. *Given an instance (C, R) and any vertex $v \in R$, if $|N^{\geq k'+1}_{G[C \cup \{v\}]}(v)| + h - |C| - 1 < k'$, the vertex v should be removed form R.*

Besides the above reduction rule, we can also move the promising vertices from R to C which exist in any solution derived from current C and R.

Reduction Rule 2. *Given an instance (C, R), if there is a $v \in C$ with $|N^{\geq k'+1}_{G[C \cup R]}(v)| = k'$, all vertices in $N^{\geq k'+1}_{G[C \cup R]}(v)$ should be moved to C.*

Algorithm 4: BranchCheck

Input : partial solution C, candidate vertex set R, size constraint $[l, h]$, current largest min-trussness k' among each computed result
Output : continue the branch or not

1 $\tau_{G[C]}(u)$ of each $u \in C$ by truss maintenance [52];
2 **for** each $u \in C$ **do**
3 $budget(u) \leftarrow min\{h - |C|, |N_G(u) \cap R|\}$;
4 $visit(u) \leftarrow False$;
5 $cost(u) \leftarrow \max\{k' + 1 - \tau_{G[C]}(u), 0\}$;
6 **if** $budget(u) < cost(u)$ **then return False** ;
7 $b_{min} \leftarrow \min_{u \in C}(budget(u))$; $c_{max} \leftarrow \max_{u \in C}(cost(u))$;
8 **if** $b_{min} \geq 2 \times c_{max}$ **then return True**;
9 **for** each $u \in C$ **do**
10 $visit(u) \leftarrow True$;
11 **if** $cost(u) > 0$ **then**
12 $A \leftarrow N_G(u) \cap R$;
13 **for** each $x \in C \setminus N_G(A)$ and $visit(x) = False$ **do**
14 $budget(x) \leftarrow \min\{h - |C| - cost(u), budget(x)\}$;
15 **if** $budget(x) < cost(x)$ **then return False** ;
16 **return True**

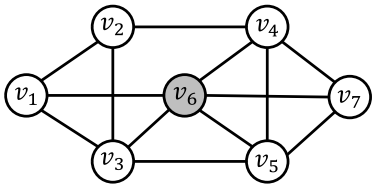


Fig. 4. Example of Backtracking

Branching Strategies. The visiting order of different branches and an effective backtracking approach also important for search efficiency. In our branching order, the selection of the vertex v^* to join C is determined by the metric in our scoring function (Definition 6), i.e., $|N^{\geq k'+1}_{G[C \cup \{v\}]}(v)| + \frac{\tau(v)}{k_{max}+1}$ for each $v \in R$.

For the backtracking, given an instance (C, R), if we find in current branch that a vertex $u \in C$ cannot exist in any qualified subgraph with min-trussness of at least $k' + 1$, the branch is unpromising. Thus, we can backtrack the above search branches until there is no such vertex. We may consider the dominance relations between different branches to early terminate the branches dominated by others, while our preliminary experiments find the speed up from the dominance relations cannot pay off the time cost of dominance check.

Example 5. *Given the graph in Figure 4, the query v_6 and size constraint $[4, 4]$, we may firstly find the subgraph induced by $\{v_1, v_3, v_5, v_6\}$ and update $k' = 3$. The backtracking approach will remove $\{v_1, v_3\}$ from C because they cannot exist in a subgraph containing v_6 with trussness $k' + 1 = 4$. Next, we continue to search with $C = \{v_5, v_6\}$ and find the qualified subgraph induced by $\{v_4, v_5, v_6, v_7\}$.*

The pseudo-code of our optimized branch and bound search, named ST-B&BP, is shown in Algorithm 5. Once we have $k' = k^*$, the subgraph H is an optimal result and is returned (Line 1). When the size of C reaches h or R becomes empty with $|C| \geq l$ (Line 2), we find the largest

k such that the size of $T^q_{G[C]}(k)$ is not less than l, denoted by $\hat{k}$ (Line 3). If $\hat{k} > k'$, the min-trussness k' is increased to $\hat{k}$ and the records H, Φ and R are updated accordingly (Lines 4-7), where Φ is a global set to record the unqualified vertices and is used for backtracking (Lines 8 and 15). If the size of C is less than h, R is not empty, current branch is promising confirmed by Algorithm 4, and C does not contain unqualified vertices (Line 8), we add vertices to C from R and continue the search (Lines 9-17). We can remove unpromising vertices from R by Reduction Rule 1 and then find the vertex v^* in R with the highest score (Lines 9-10). As v^* will be added to C, some other vertices of R may be moved together with v^* to C by Reduction Rule 2, and the set of above vertices is denoted by V^* (Lines 11-12). If V^* is empty, we return the branch (Line 13). The branch to expand C is invoked at Line 14, and Φ is recovered by $\Phi \setminus V^*$ after the return of Line 15 such that the unqualified vertices found at Line 6 of previous recursion can be removed from Φ. Then, the branch to discard V^* is invoked at Line 16 to search other promising branches. When the algorithm terminates, it finds an optimal result on C and R.

The final exact search algorithm, named ST-Exa, is given in Algorithm 6. Firstly, our advanced heuristic is executed to find a high-quality initial result H (Line 1). If the min-trussness of H is the same as the upper bound of optimal min-trussness, we return H as an optimal result; Otherwise, we initialize C and R at Line 4, and the branch and bound search is invoked at Line 5 to find an optimal result.

4.3 Extensions of the Algorithms

Finding More Optimal Communities. First, we need to compute the optimal min-trussness k^{opt} by executing Algorithm 6 and use k^{opt} to facilitate the efficient enumeration of optimal results. Note that, without the optimal k^{opt} value, Algorithm 6 has to search for all the possible results with min-trussness equal to k' (current largest min-trussness of all the computed results) until there is a result with min-trussness larger than k'. This will cause redundant computations once k' can be larger. Thus, it is more efficient to first compute the optimal min-trussness k^{opt}.

Then, the idea is to enumerate the optimal results on the k^{opt}-truss. Specifically, we remove Lines 1-3 of Algorithm 6, remove every k^* in the algorithms, replace k' with $k^{opt} - 1$ in Lines 4-5 of Algorithm 6, remove Line 1 of Algorithm 5, update Line 5 of Algorithm 5 to "$H \leftarrow T^q_{G[C]}(k^{opt})$;" and replace "**return** H" with "**output** H and **return**" in Lines 13 and 17 of Algorithm 5. Note that our updated algorithms will only output every "maximal" optimal result H, i.e., there is no other optimal result containing H, which will not be overwhelming to end users.

Execution with a Proper Size Constraint. For the applications with clear size constraints (decided by the corresponding budget or capacity), we simply execute the proposed algorithm with the given constraints. For applications without clear size constraints, we may adjust the size upper bound to search for communities with different sizes, because our algorithms prefer to search the results with sizes close to the upper bound.

Algorithm 5: ST-B&BP

Input : a graph G, the trussness of each edge, size constraint [l,h], query vertex q, the upper bound of optimal min-trussness k^*, current largest trussness k' among the min-trussness of each computed result, current partial set C, candidate vertex set R
Output : an optimal result H on the subgraph induced by C and R

1 **if** $k' = k^*$ **then return** H;
2 **if** $|C| = h$ or ($|C| \geq l$ and $|R| = 0$) **then**
3 $\quad \hat{k} \leftarrow \max\{k \mid |V(T^q_{G[C]}(k))| \geq l\}$;
4 $\quad$ **if** $\hat{k} > k'$ **then**
5 $\quad\quad k' \leftarrow \hat{k}$; $H \leftarrow T^q_{G[C]}(k')$;
6 $\quad\quad \Phi \leftarrow \{v \mid v \notin T^q(k'+1) \wedge v \in C\}$;
7 $\quad\quad R \leftarrow R \setminus \{v \mid \tau(v) \leq k' \wedge v \in R\}$;
8 **if** $|C| < h$ and $|R| \neq 0$ and BranchCheck(C, R, l, h, k') and $C \cap \Phi = \emptyset$ **then**
9 $\quad$ Update R by Reduction Rule 1;
10 $\quad v^* \leftarrow$ the vertex in R with the highest score;
11 $\quad V' \leftarrow$ the set of vertices to be moved from R to C by checking v^* with Reduction Rule 2;
12 $\quad V^* \leftarrow \{v^*\} \cup V'$;
13 $\quad$ **if** $V^* = \emptyset$ **then return** H;
14 $\quad$ **ST-B&BP**$(..., C \cup V^*, R \setminus V^*)$;
15 $\quad \Phi \leftarrow \Phi \setminus V^*$;
16 $\quad$ **ST-B&BP**$(..., C, R \setminus V^*)$;
17 **return** H

Algorithm 6: ST-Exa

Input : a graph G, the trussness of each edge, size constraint [l,h], query vertex q
Output : an optimal result H

1 $\{H, k^*\} \leftarrow$ **ST-Heu**(G, τ, l, h, q);
2 $k' \leftarrow$ the minimum vertex trussness of H;
3 **if** $k' \neq k^*$ **then**
4 $\quad C \leftarrow \{q\}$; $R \leftarrow N^{\geq k'+1}(C)$;
5 $\quad H \leftarrow$ **ST-B&BP**$(G, \tau, l, h, q, k^*, k', C, R)$;
6 **return** H

5 Experimental Evaluation

In this section, extensive experiments are conducted to verify the performance of our algorithms. Here we briefly summarize the results. For model effectiveness, our ST-Exa algorithm can well constrain the community size (Exp-1) and outperform the state-of-the-art in terms of community scores (Exp-2) which is also verified by our case study on DBLP dataset (Exp-3). For algorithm performance, the result (cohesiveness) of ST-Exa within a time limit is close to the upper bound cohesiveness of the exact result (Exp-4). The outperformance is similar when the size constraint is large (Exp-5) and querying on different vertices (Exp-6). Though the k-truss computation is more costly than k-core, our well-designed ST-Exa is more efficient than its competitor (Exp-7). Our proposed techniques can also be used to improve the efficiency of the SC-BRB algorithm (Exp-8). We also evaluate the performance of different heuristics (Exp-9) and the effectiveness of each proposed technique (Exp-10).

TABLE 2
Statistics of Datasets

Dataset	n	m	d_{avg}	d_{max}	k_{max}	$\|T(5)\|/\|V\|$
Email	36K	183K	10.02	1383	22	0.39
Hepph	34K	421K	24.36	846	25	0.66
Epinions	131K	841K	3.68	96K	43	0.08
DBLP	317K	1.0M	6.62	343	114	0.40
Flickr	105K	2.3M	43.74	5K	574	0.16
Google	875K	4.3M	9.87	6K	44	0.46
Youtube	1.1M	3.0M	5.26	28K	19	0.04
Berkstan	686K	6.6M	19.4	84K	201	0.57
Orkut	3.1M	117.1M	76.3	33K	78	0.79
Wiki	6M	142.6M	28.33	195K	71	0.62
UK	18.4M	261.6M	28.33	194K	944	0.62
Webbase	118.1M	1019.9M	17.26	816K	1507	0.41

5.1 Experimental Setting

Datasets. We use 12 public real-life networks in our experiments. Wiki2020, Uk2002 and Webbase are from Webgraph[1] and the others are from SNAP[2]. The details of the networks are shown in Table 2, where d_{avg}, d_{max} and k_{max} denote the average degree, maximum degree and the largest trussness value. The last column records the ratio of the vertex set size in 5-truss over the whole vertex set. The abbreviation of each dataset is marked in bold.

Algorithms. Our main competitors are the algorithms in [10], including the heuristic SC-Heu and the exact algorithm SC-BRB. We also report some results of SCkT in [11]. The codes of SCkT, SC-Heu, and SC-BRB are kindly provided by the authors. The algorithms are summarized as follows.

- SC-Heu and SC-BRB: The heuristic and exact algorithms are proposed in [10], respectively, target at finding a subgraph with the largest min-degree among every connected subgraph containing query vertex q that has at least l and at most h vertices. We also evaluate SC-BRB* which is SC-BRB equipped with our proposed techniques. The details are given in Exp-8.
- SCkT: the exact search algorithm in [11] for searching a triangle-connected k-truss subgraph containing query vertex q with size not exceeding h, where k is user-specified. Note that the query on a vertex with trussness less than k will return an empty result. The percentage of non-empty queries for SCkT with $k = 5$ is at most $|T(5)|/|V|$ shown in Table 2.
- ST-Heu, ST-Cli, ST-Base and ST-BaseT: ST-Heu is our advanced heuristic (Algorithm 2). Its degraded versions are ST-Base (Algorithm 1) and ST-Cli (Section 4.1), respectively. ST-BaseT uses $Score(C, R) = \arg\max_{v \in R}\{v \mid \triangle(v) + \frac{\tau(v)}{k_{max}+1}\}$ to guide vertex selection where $\triangle(v)$ is the number of triangles containing v in $G[C \cup \{v\}]$, which is the only difference to ST-Base.
- ST-Exa: Our exact search algorithm (Algorithm 6). ST-Exa\H is the ST-Exa that starts with k' = 1 and without updating k' by ST-Heu. Similarly, ST-Exa\R, ST-Exa\L and ST-Exa\B are ST-Exa without Reduction Rules, Budget-Cost Bounding and Backtracking Approach (Section 4.2), respectively.

Note that, if a test cannot finish within the time limit, we return the best qualified subgraph which satisfies the connectivity and size constraint while the min-support from ST-Exa (resp. min-degree from SC-BRB) may not be optimal.

1. https://webgraph.di.unimi.it/
2. http://snap.stanford.edu/

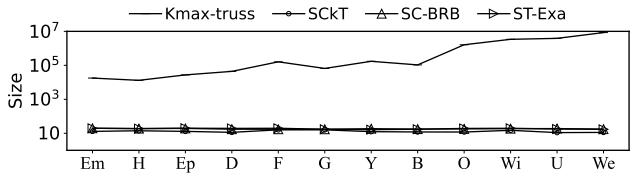


Fig. 5. Average Result Size with Size Constraint $[11, 20]$

Parameters. All experiments are conducted with time limit of 1000 seconds. Note that the performance gain is minor on hard cases with a larger time limit. The default size constraint is $[11, 20]$, i.e., the size of the returned subgraph is at least 11 and at most 20. As the sizes of ground-truth communities are diverse, we also report the results on larger size ranges and/or larger community sizes. For each algorithm, as querying on 2-truss vertices is trivial, we report the average result of 200 independent tests by querying random vertices in 3-trusses. Note that in each test the query vertices for different algorithms are the same.

Environment. All algorithms are implemented in C++ and compiled by GCC (7.5.0) under O3 optimization. We conduct all experiments on a machine with an Intel Xeon 2.1GHz CPU and 512G main memory.

Our source code is shared online [3].

5.2 Evaluation on Effectiveness of Models

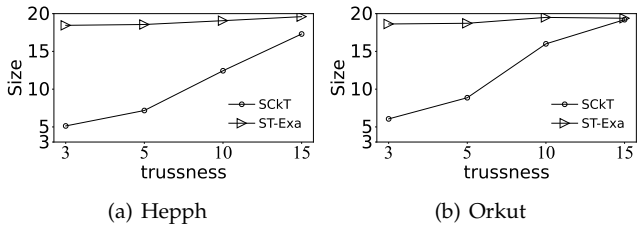
In this subsection, we report the result sizes, the scores on two community metrics, and the case study.

Exp-1: Result Size. Figure 5 shows the average number of vertices in the returned subgraph for each algorithm. Kmax-truss in the figure represents the average size on all the k_{max}-trussness. For SCkT, we set $k = 10$ and report the result by querying each vertex with trussness of at least 10 (for returning non-empty results). The result size is well constrained for each algorithm except Kmax-truss. It is observed that the result size is rarely affected by the scale of dataset. The result size of SCkT is close to 10 while the result sizes of ST-Exa and SC-BRB are close to 20, because of the different search preferences. The result size of ST-Exa is larger than SC-BRB by 0.5 on average, because ST-Exa returns a result when the size of C is close to h while SC-BRB returns when the size of C is no less than l.

We also report the result sizes by varying the parameter k of SCkT. Note that ST-Exa is parameter-free on k. Figure 6 shows the result size from SCkT is close to k while the size from ST-Exa is more stable, closing to the size constraint h.

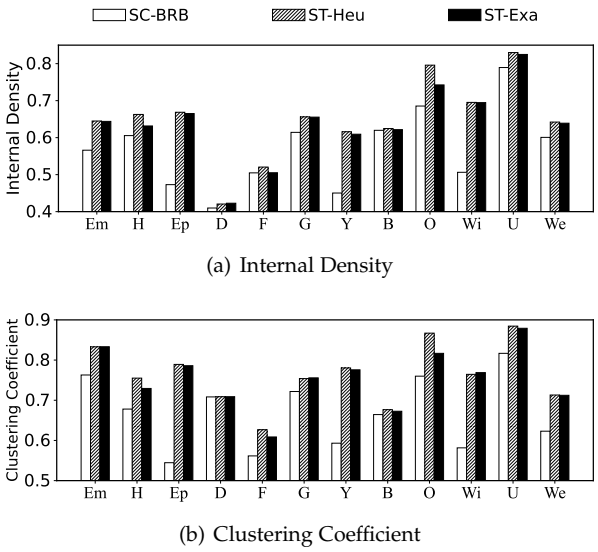
Exp-2: Results on Two Community Metrics. We adopt two representative metrics to measure the quality of the returned subgraph, i.e., Internal Density ($\frac{2m}{n \times (n-1)}$) [53] and Clustering Coefficient ($\frac{3 \times |\triangle|}{|triplet|}$) [54] where a higher score is preferred. As shown in Figure 7, our ST-Heu and ST-Exa significantly outperform SC-BRB in terms of the two community metrics. The values of clustering coefficient are similar on DBLP because the authors in DBLP are usually

3. https://github.com/codecreateworld321/STCS



(a) Hepph (b) Orkut

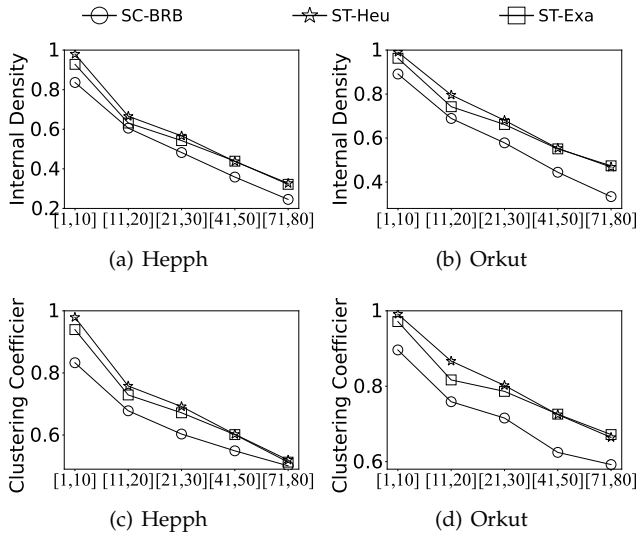
Fig. 6. Result Size by Varying k for SCkT



(a) Internal Density

(b) Clustering Coefficient

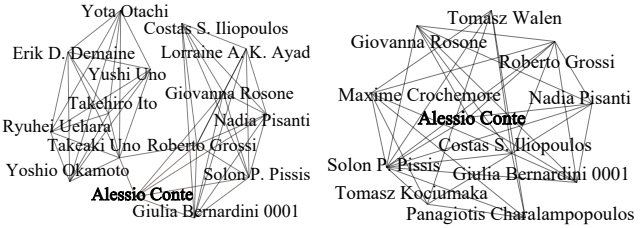
Fig. 7. Result Quality with Size Constraint $[11, 20]$



(a) Hepph (b) Orkut

(c) Hepph (d) Orkut

Fig. 8. Community Scores by Varying Size Constraints



(a) SCS (b) STCS

Fig. 9. Case Study on DBLP

in small and dense groups. It is interesting to see the scores of ST-Heu can be slightly higher than ST-Exa on the two community metrics. This is because ST-Heu prefers to include close vertices with large degrees, corresponding to better internal density and clustering coefficient, while the min-trussness (our optimization objective for ST-Exa) is used to find well-connected higher-order structures.

Note that, in our quality evaluation (Exp-4, Figure 10), ST-Exa largely outperforms ST-Heu regarding success ratio and the min-trussness. Thus, the two community metrics cannot be used to replace the min-trussness metric and the optimization on min-trussness reveals a new angle of dense structures. For instance, in our case study (Exp-3, Figure 9), the result of SCS has a higher clustering coefficient than STCS, while the result of STCS is clearly better connected than SCS, corresponding to a larger min-trussness.

We also compare the result qualities of SC-BRB, ST-Heu, and ST-Exa by varying the size constraint on Hepph and Orkut datasets. There are five different size constraint settings where $[l, h]$ are varied from $[1, 10]$ to $[71, 80]$. As shown in Figure 8, our algorithms outperform SC-BRB algorithm on all the settings.

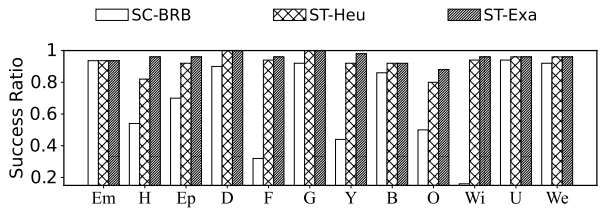
Exp-3: Case Study on DBLP. Figure 9 depicts the results of SC-BRB and ST-Exa, respectively, by querying the scholar "Alessio Conte" on the DBLP dataset from 2000 to 2022 with size constraint $[10, 15]$. In the case study, there is an edge between two authors iff the number of their co-authored papers is at least 5. Figure 9(a) shows the optimal result of SCS returned by SC-BRB, with minimum degree 6 and size 12. Figure 9(b) shows the optimal result of STCS returned by ST-Exa, which is more well-connected because the minimum vertex trussness is 6 for STCS and 3 for SCS, respectively.

We find that the average number of co-authored papers between two authors in STCS is 25.4, which is higher than SCS (18.3). Besides, there are two distinct groups in SCS, with a weak connection between them. Moreover, the diameter of STCS is 2, while the diameter of SCS is 3.

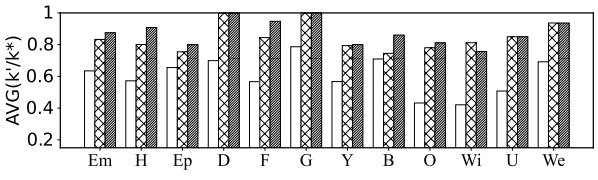
The reason for the above gaps is that SCS pursues the vertices with large degrees while neglecting the higher-order structures among the authors considered by STCS. Therefore, the STCS problem based on min-trussness is more promising for size-constrained community search.

5.3 Evaluation on Performances of Algorithms

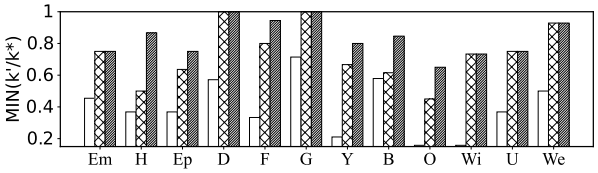
In this subsection, we first report success ratio within the time limit of 1000s, where the **success ratio** is the ratio that a query result is confirmed to be optimal, i.e., the min-trussness of the resulting subgraph k' is equal to the min-trussness upper bound k^* for ST-Heu and ST-Exa. Note that the true success ratio is higher than the above ratio, while it is costly to compute the optimal min-trussness in some cases. Correspondingly, for SC-BRB, we set k^* by the largest min-degree of a connected k-core including query vertex q with size no less than l. Then, we evaluate the querying on different cases and the efficiency of the algorithms including the running time of SC-BRB optimized by our techniques.



(a) Success Ratio



(b) Average of k'/k^* on Hard Cases ($k' \neq k^*$)



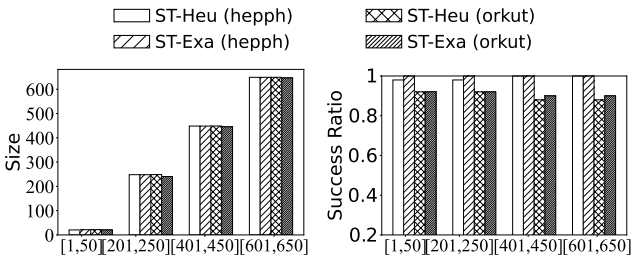
(c) Minimum of k'/k^* on Hard Cases ($k' \neq k^*$)

Fig. 10. Success Ratio and Approximation to k^* on All Datasets

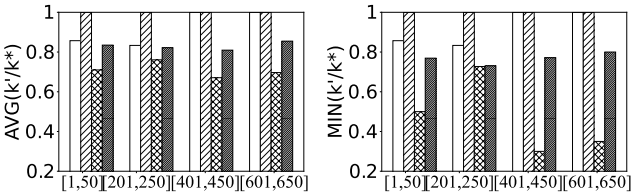
Exp-4: Success Ratio and Approximation to k^*. Figure 10(a) shows the performance quality of 3 algorithms regarding success ratio and approximation to k^*. Our ST-Exa largely outperforms SC-BRB on success ratio where the gap is up to 5 times on Wiki benefiting from our well-designed heuristic ST-Heu. As for the performance on hard cases, Figures 10(b) and 10(c) report the average values and the minimum values of k'/k^* on hard cases, respectively. Here a hard case is that the min-trussness of the resulting subgraph returned within 1000s does not equal to the min-trussness upper bound, i.e., $k' \neq k^*$. The results of ST-Heu on hard cases are better than ST-Exa on Wiki dataset because their hard cases are different queries and some hard cases of ST-Heu can be optimally addressed by ST-Exa as shown in Figure 10(a). Overall, Figure 10 shows our ST-Exa returns an optimal result in over 88% cases and the results on hard cases are close to the optimal values, where the min-trussness is at least 65% and on average 74% of the optimal value.

Exp-5: Querying on Large Size Constraints. Figure 11 shows the performance of ST-Heu and ST-Exa on Hepph and Orkut with the size constraints on larger ranges, i.e., $[1, 50]$, $[201, 250]$, $[401, 450]$ and $[601, 650]$. Figure 11(a) shows the result sizes returned by our ST-Heu and ST-Exa are close to the upper bound of the size limit. The result size from ST-Heu is slightly larger than ST-Exa, because ST-Exa pursues the results with higher min-trussnesses than ST-Heu and these results may be smaller.

Figure 11(b) shows that the success ratios of our algorithms are at least 90% on Hepph and Orkut, for different constraints. Besides, we find ST-Heu and ST-Exa have better success ratios on constraint $[1, 50]$ than $[11, 20]$ (Figure 10(a)) on Hepph and Orkut. This is because the queries with large size ranges are more tractable to find the exact result. Figure

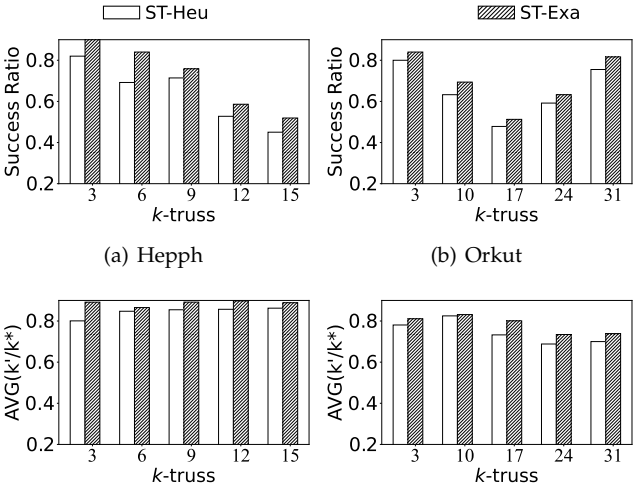


(a) Size Constraint (b) Success Ratio



(c) Avg(k'/k^*) on Hard Case (d) Min(k'/k^*) on Hard Case

Fig. 11. Querying on Large Size Constraints



(a) Hepph (b) Orkut

(c) Hepph (k'/k^* on Hard Cases) (d) Orkut (k'/k^* on Hard Cases)

Fig. 12. Success Ratio and Approximation to k^* on Different k-Trusses

11(c) shows the averages of k'/k^* of our algorithms on hard cases (the cases with $k' \neq k^*$) are more than 67% for all the size constraints. Figure 11(d) shows the gaps of the minimums of $k'/k*$ between ST-Heu and ST-Exa are large on some of the hard cases. Thus, the search in ST-Exa is necessary when the heuristic is unable to find a result close to the optimal.

Exp-6: Querying on Different k-Trusses. Figure 12 reports the success ratios and the averages of k'/k^* of our algorithms by querying on random vertices in k-truss with different k. For each k value, we conduct 50 independent tests and each test queries a random vertex in k-truss of G. The values of k for Hepph datasets are from 3 to 15, where the size ratio of 15-truss vertex set over the whole vertex set is 2.5%. The k values of Orkut are from 3 to 31, where the size ratio of 31-truss vertex set over the whole vertex set is 2.3%. Figure 12 reveals that more optimal results can be found when k is relatively large or small compared with the size constraint. The hard cases are mainly on querying

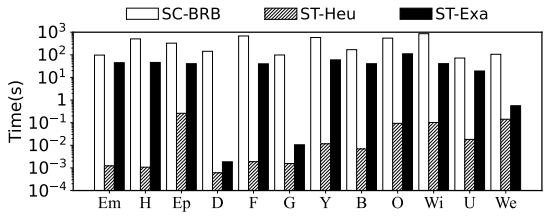


Fig. 13. Running Time on All Datasets

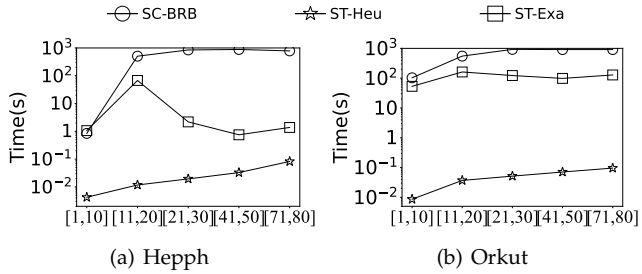


Fig. 14. Running Time with Different Size Constraints

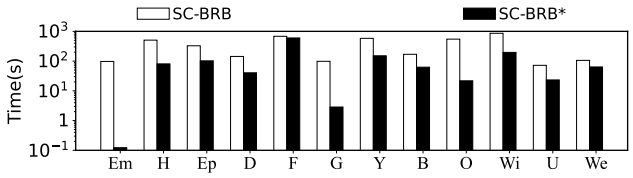


Fig. 15. Running Time of SC-BRB* equipped with Our Techniques

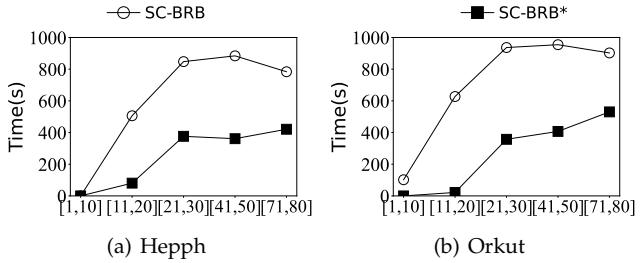


Fig. 16. Running Time of SC-BRB* with Different Size Constraints

the vertices with trussness between $l = 11$ and $h = 20$. Although the success ratio with a hard k value may not be large as shown in Figures 12(a-b), the non-optimal results returned by our algorithm are of high-quality regarding the cohesiveness objective as shown in Figures 12(c-d).

Exp-7: Running Time of the Algorithms. Figure 13 reports the time cost of ST-Heu, ST-Exa, and SC-BRB, respectively. It shows that our ST-Heu and ST-Exa outperform SC-BRB on all datasets where the speed up is up to 5 orders of magnitude. The main reason is because that our heuristic can quickly find high-quality initial result and our optimizations are effective. ST-Exa is slower than ST-Heu because it contains ST-Heu and conducts the exact search. The gap between ST-Heu and ST-Exa is small on DBLP or Google because ST-Heu often finds the optimal results.

We also evaluate the running time under different size constraints. The size constraint $[l, h]$ is varied from $[1, 10]$ to $[71, 80]$. Figure 14 shows our algorithms are faster than SC-BRB on all the settings. For ST-Heu and SC-BRB, the time cost becomes larger with the increase of size constraint due to larger search space. ST-Exa is different because its running time is influenced by the success ratio of ST-Heu. When size constraint increases, the success ratio of ST-Heu may be improved and thus the time cost of ST-Exa may become smaller.

Exp-8: Running Time of SC-BRB with Our Techniques. To further validate the effectiveness of our techniques, we evaluate SC-BRB* which is SC-BRB equipped with some of our proposed techniques. SC-BRB* can be easily implemented by replacing some functions of SC-BRB as follows. Firstly, all the operations on "trussness" in our techniques are replaced by "coreness", e.g., k_{max} in Scoring Function (Definition 6) is set by the largest vertex coreness in G. Then, the heuristic used in SC-BRB is replaced by ST-Heu adapted on "coreness". Finally, if there is a vertex in C with its coreness in G no larger than k', SC-BRB* terminates current branch with our Backtracking technique. Note that SC-BRB* do not contain our Reduction Rules and Budget-Cost Bounding as they are designed for handling support values rather than degree and cannot be well adapted. Figure 15 shows SC-BRB* outperforms SC-BRB on all datasets, where the speed up is up to three orders of magnitude on Email due to the effective heuristic. In Figure 16, SC-BRB* still has a better performance with different size constraints.

5.4 Evaluation on Optimization Techniques

In this subsection, we validate the effectiveness of each proposed technique.

Exp-9: Performance of Different Heuristics. Table 3 shows that the success ratio of ST-Heu outperforms other heuristics and it is at least 14% higher than SC-Heu proposed in [10]. When the success ratio is close to 1, it is hard to further improve the ratio, while ST-Heu still improves the ratio of ST-Cli from 0.84 to 0.94 on Wiki due to the novel design. In Figure 17, the time costs of ST-Heu are not large on all the datasets as its framework is light-weight. The runtime of ST-Heu is less than 1s on every dataset. The success ratio of ST-BaseT is similar to ST-Base while its time cost is larger due to triangle counting. Overall, the results in Table 3 and Figure 17 show that ST-Heu can well initialize a result for STCS, because it addresses "slow start" and "branch trap" issues as discussed in Section 4.1.

We also evaluate the effect of using different diversified cliques in ST-Heu. The result shows that the success ratio from diversified top-2 cliques is at least 97% of the success ratio by using all the maximal cliques in ST-Heu. Thus, we use diversified top-2 cliques in ST-Heu for a good performance trade-off.

Exp-10: Performance of Optimization Techniques. Table 4 shows the runtime of ST-Exa by unloading each proposed technique. As ST-Exa achieves the smallest time cost on every setting, all the optimization techniques proposed in the paper are effective. The performance gap between ST-Exa$\backslash$H and ST-Exa is the largest, because the heuristic is critical for improving the efficiency of exact search.

TABLE 3
Success Ratio on Heuristic Algorithms

Dataset	SC-Heu	ST-BaseT	ST-Base	ST-Cli	ST-Heu
Em	0.68	**0.94**	0.9	0.91	**0.94**
H	0.3	0.8	0.66	0.78	**0.82**
Ep	0.66	**0.92**	**0.92**	**0.92**	**0.92**
D	0.54	0.9	**1**	**1**	**1**
F	0.4	0.84	0.9	**0.94**	**0.94**
G	0.86	0.96	0.98	**1**	**1**
Y	0.4	0.9	0.9	**0.92**	**0.92**
B	0.68	**0.92**	0.88	**0.92**	**0.92**
O	0.34	0.72	0.74	0.74	**0.8**
Wi	0.3	0.86	0.72	0.84	**0.94**
U	0.8	0.9	0.92	**0.96**	**0.96**
We	0.84	0.92	**0.96**	**0.96**	**0.96**

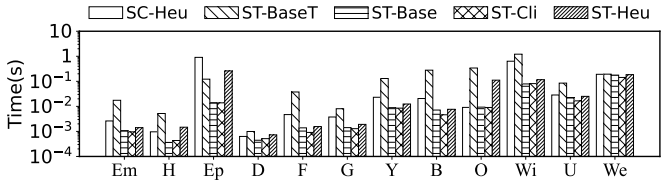


Fig. 17. Running Time on Heuristic Algorithms

TABLE 4
Running time on Exact Algorithms

Dataset	ST-Exa\B	ST-Exa\L	ST-Exa\R	ST-Exa\H	ST-Exa
Em	65.54	69.43	66.01	92.54	**63.83**
H	51.56	160.06	60.43	138.82	**45.72**
Ep	40.46	80.02	42.21	142.94	**40.37**
D	**0.001**	**0.001**	**0.001**	20.02	**0.001**
F	**40.01**	60.01	**40.01**	87.88	**40.01**
G	**0.003**	0.004	**0.003**	40.03	**0.003**
Y	62.55	80.08	80.07	103.52	**59.74**
B	**40.35**	41.02	**40.35**	66.22	**40.35**
O	111.81	186.87	121.13	357.75	**110.57**
Wi	**60.41**	100.09	73.86	291.34	**60.41**
U	**19.28**	40.03	20.03	73.21	**19.28**
We	**0.45**	40.18	1.42	21.44	**0.45**

6 Conclusion

Motivated by the defects in existing solutions, we study the STCS problem for size-constrained community search. We prove that the STCS problem is NP-hard and hard to approximate. We propose a branch and bound algorithm ST-Exa to find an optimal result, equipped with effective optimizations. A novel heuristic is designed to find a promising initial result and reduce the search space. Extensive experiments on 12 real-world graphs show that the quality of the query result from ST-Exa is better and ST-Exa is faster by up to 5 orders of magnitude, compared with the state-of-the-art. As the size of a community may relate to its properties [55], it is interesting to find a proper size constraint automatically in future studies. As real communities may be diverse, it is also interesting to find diversified results on size-constrained communities.

Acknowledgments

This work is partially supported by the National Natural Science Foundation of China (U20B2046, 62002073) and the Guangdong Basic and Applied Basic Research Foundation (2023A1515012603, 2019B1515120048).

References

[1] Y. Zhang, L. Qin, F. Zhang, and W. Zhang, "Hierarchical decomposition of big graphs," in *ICDE*, pp. 2064–2067, IEEE, 2019.
[2] Y. Fang, X. Huang, L. Qin, Y. Zhang, W. Zhang, R. Cheng, and X. Lin, "A survey of community search over big graphs," *VLDB J.*, vol. 29, no. 1, pp. 353–392, 2020.
[3] J. Zhang, P. S. Yu, and Y. Lv, "Enterprise employee training via project team formation," in *WSDM*, pp. 3–12, 2017.
[4] J. She, Y. Tong, L. Chen, and T. Song, "Feedback-aware social event-participant arrangement," in *SIGMOD*, pp. 851–865, ACM, 2017.
[5] J. Ugander, L. Backstrom, C. Marlow, and J. M. Kleinberg, "Structural diversity in social contagion," *Proc. Natl. Acad. Sci. USA*, vol. 109, no. 16, pp. 5962–5966, 2012.
[6] P. Manchanda, G. Packard, and A. Pattabhiramaiah, "Social dollars: The economic impact of customer participation in a firm-sponsored online customer community," *Mark. Sci.*, vol. 34, no. 3, pp. 367–387, 2015.
[7] M. T. Dittrich, G. W. Klau, A. Rosenwald, T. Dandekar, and T. Müller, "Identifying functional modules in protein–protein interaction networks: an integrated exact approach," *Bioinformatics*, vol. 24, no. 13, pp. i223–i231, 2008.
[8] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *SIGKDD*, pp. 939–948, 2010.
[9] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, "Efficient and effective community search," *Data Mining and Knowledge Discovery*, vol. 29, pp. 1406–1433, Sep 2015.
[10] K. Yao and L. Chang, "Efficient size-bounded community search over large networks," *PVLDB*, vol. 14, no. 8, pp. 1441–1453, 2021.
[11] B. Liu, F. Zhang, W. Zhang, X. Lin, and Y. Zhang, "Efficient community search with size constraint," in *ICDE*, pp. 97–108, IEEE, 2021.
[12] J. Wang and J. Cheng, "Truss decomposition in massive networks," *PVLDB*, vol. 5, no. 9, pp. 812–823, 2012.
[13] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu, "Querying k-truss community in large and dynamic graphs," in *SIGMOD*, pp. 1311–1322, 2014.
[14] E. Akbas and P. Zhao, "Truss-based community search: a truss-equivalence based indexing approach," *PVLDB*, vol. 10, no. 11, pp. 1298–1309, 2017.
[15] S. Wasserman and K. Faust, *Social network analysis: Methods and applications*, vol. 8. Cambridge university press, 1994.
[16] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, p. 440, 1998.
[17] V. Batagelj and M. Zaveršnik, "Short cycle connectivity," *Discrete Mathematics*, vol. 307, no. 3-5, pp. 310–318, 2007.
[18] J. Cohen, "Trusses: Cohesive subgraphs for social network analysis," *National security agency technical report*, vol. 16, pp. 3–1, 2008.
[19] C. Bron and J. Kerbosch, "Finding all cliques of an undirected graph (algorithm 457)," *Commun. ACM*, vol. 16, no. 9, pp. 575–576, 1973.
[20] J. Cheng, Y. Ke, A. W. Fu, J. X. Yu, and L. Zhu, "Finding maximal cliques in massive networks by h*-graph," in *SIGMOD*, pp. 447–458, 2010.
[21] J. Abello, M. G. C. Resende, and S. Sudarsky, "Massive quasi-clique detection," in *LATIN*, pp. 598–612, 2002.
[22] J. Pei, D. Jiang, and A. Zhang, "On mining cross-graph quasi-cliques," in *KDD*, 2005.
[23] F. Bonchi, A. Khan, and L. Severini, "Distance-generalized core decomposition," in *SIGMOD*, pp. 1006–1023.
[24] C. Giatsidis, F. Malliaros, D. M. Thilikos, and M. Vazirgiannis, "Corecluster: A degeneracy based graph clustering framework," in *IAAI*, 2014.
[25] D. W. Matula and L. L. Beck, "Smallest-last ordering and clustering and graph coloring algorithms," *J. ACM*, vol. 30, no. 3, pp. 417–427, 1983.
[26] S. B. Seidman, "Network structure and minimum degree," *Social Networks*, vol. 5, no. 3, pp. 269–287, 1983.
[27] B. Liu, F. Zhang, C. Zhang, W. Zhang, and X. Lin, "Corecube: Core decomposition in multilayer graphs," in *WISE*, vol. 11881, pp. 694–710, Springer, 2019.
[28] Y. Shao, L. Chen, and B. Cui, "Efficient cohesive subgraphs detection in parallel," in *SIGMOD*, pp. 613–624, 2014.

[29] 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*, pp. 205–216, 2013.

[30] R. Zhou, C. Liu, J. X. Yu, W. Liang, B. Chen, and J. Li, "Finding maximal k-edge-connected subgraphs from a large graph," in *EDBT*, pp. 480–491, 2012.

[31] S. Sahu, A. Mhedhbi, S. Salihoglu, J. Lin, and M. T. Özsu, "The ubiquity of large graphs and surprising challenges of graph processing," *PVLDB*, vol. 11, no. 4, pp. 420–431, 2017.

[32] F. D. Malliaros, C. Giatsidis, A. N. Papadopoulos, and M. Vazirgiannis, "The core decomposition of networks: theory, algorithms and applications," *VLDB J.*, vol. 29, no. 1, pp. 61–92, 2020.

[33] V. E. Lee, N. Ruan, R. Jin, and C. Aggarwal, "A survey of algorithms for dense subgraph discovery," in *Managing and Mining Graph Data*, pp. 303–336, Springer, 2010.

[34] X. Jian, Y. Wang, and L. Chen, "Effective and efficient relational community detection and search in large dynamic heterogeneous information networks," *PVLDB*, vol. 13, no. 10, pp. 1723–1736, 2020.

[35] Y. Fang, Y. Yang, W. Zhang, X. Lin, and X. Cao, "Effective and efficient community search over large heterogeneous information networks," *PVLDB*, vol. 13, no. 6, pp. 854–867, 2020.

[36] B. Ghosh, M. E. Ali, F. M. Choudhury, S. H. Apon, T. Sellis, and J. Li, "The flexible socio spatial group queries," *PVLDB*, vol. 12, no. 2, pp. 99–111, 2018.

[37] X. Huang and L. V. S. Lakshmanan, "Attribute-driven community search," *PVLDB*, vol. 10, no. 9, pp. 949–960, 2017.

[38] X. Huang, L. V. S. Lakshmanan, J. X. Yu, and H. Cheng, "Approximate closest community search in networks," *PVLDB*, vol. 9, no. 4, pp. 276–287, 2015.

[39] Z. Zheng, F. Ye, R.-H. Li, G. Ling, and T. Jin, "Finding weighted k-truss communities in large networks," *Information Sciences*, vol. 417, pp. 344–360, 2017.

[40] K. Wang, X. Lin, L. Qin, W. Zhang, and Y. Zhang, "Vertex priority based butterfly counting for large-scale bipartite networks," *PVLDB*, vol. 12, no. 10, pp. 1139–1152, 2019.

[41] B. Liu, L. Yuan, X. Lin, L. Qin, W. Zhang, and J. Zhou, "Efficient (α, β)-core computation in bipartite graphs," *VLDB J.*, vol. 29, no. 5, pp. 1075–1099, 2020.

[42] Q. Liu, M. Zhao, X. Huang, J. Xu, and Y. Gao, "Truss-based community search over large directed graphs," in *SIGMOD*, pp. 2183–2197, 2020.

[43] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo, "Efficient and effective community search," *Data mining and knowledge discovery*, vol. 29, no. 5, pp. 1406–1433, 2015.

[44] Y. Ma, Y. Yuan, F. Zhu, G. Wang, J. Xiao, and J. Wang, "Who should be invited to my party: A size-constraint k-core problem in social networks," *J. Comput. Sci. Technol*, vol. 34, no. 1, pp. 170–184, 2019.

[45] W. Cui, Y. Xiao, H. Wang, and W. Wang, "Local search of communities in large graphs," in *SIGMOD*, pp. 991–1002, 2014.

[46] C. Li, F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin, "Efficient progressive minimum k-core search," *PVLDB*, vol. 13, no. 3, pp. 362–375, 2019.

[47] X. Huang, W. Lu, and L. V. S. Lakshmanan, "Truss decomposition of probabilistic graphs: Semantics and algorithms," in *SIGMOD*, pp. 77–90, 2016.

[48] F. Zhang, L. Yuan, Y. Zhang, L. Qin, X. Lin, and A. Zhou, "Discovering strong communities with user engagement and tie strength," in *DASFAA*, pp. 425–441, 2018.

[49] M. S. Granovetter, "The strength of weak ties," *American journal of sociology*, vol. 78, no. 6, pp. 1360–1380, 1973.

[50] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[51] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang, "Diversified top-k clique search," *VLDB J.*, vol. 25, no. 2, pp. 171–196, 2016.

[52] Y. Zhang and J. X. Yu, "Unboundedness and efficiency of truss maintenance in evolving graphs," in *SIGMOD*, pp. 1024–1041.

[53] T. F. Coleman and J. J. Moré, "Estimation of sparse jacobian matrices and graph coloring blems," *SIAM journal on Numerical Analysis*, vol. 20, no. 1, pp. 187–209, 1983.

[54] R. D. Luce and A. D. Perry, "A method of matrix analysis of group structure," *Psychometrika*, vol. 14, no. 2, pp. 95–116, 1949.

[55] J. Leskovec, K. J. Lang, A. Dasgupta, and M. W. Mahoney, "Statistical properties of community structure in large social and information networks," in *WWW*, pp. 695–704, 2008.

Fan Zhang is a Professor at Guangzhou University. He was a research associate in the School of Computer Science and Engineering, University of New South Wales, from 2017 to 2019. He received the BEng degree from Zhejiang University in 2014, and the PhD from University of Technology Sydney in 2017. His research interests include graph algorithms and social networks. Since 2017, he has published more than 20 papers in top venues, e.g., SIGMOD, PVLDB, ICDE, IJCAI, AAAI, TKDE and VLDB Journal.

Haicheng Guo is currently working towards the M.E. degree in the Cyberspace Institute of Advanced Technology of Guangzhou University, Guangzhou, China. His received the B.Eng. degree from the School of Computer Science and Engineering of Central South University, Changsha, China in 2019. His research interests include graph algorithms and social networks.

Dian Ouyang is an Associate Professor at Guangzhou University. He was a research associate in the University of Sydney from 2019 to 2020. He received the BEng degree from Renmin University of China in 2015, and the PhD from University of Technology Sydney in 2019. His research interests include graph algorithms and query optimisation. He has published several papers in top conferences including VLDB and SIGMOD.

Shiyu Yang is a Professor at Cyberspace Institute of Advanced Technology, Guangzhou University. He received the Ph.D. degree from the University of New South Wales, Sydney, Australia in 2016. He received his BSc and MSc degrees in Computer Science from Dalian University of Technology, China. His research interests include spatiotemporal database, data mining and query optimization. He has published papers in conferences and journals including ICDE, PVLDB and VLDB Journal.

Xuemin Lin received the BSc degree in applied math from Fudan University, in 1984, and the PhD degree in computer science from the University of Queensland, in 1992. He is currently a chair professor in Shanghai Jiao Tong University. He was a scientia professor and the head of the Database Research Group, with the School of Computer Science and Engineering, University of New South Wales. He was the editor-in-chief of the IEEE Transactions on Knowledge and Data Engineering. He is an IEEE Fellow. His current research interests lie in Graph Mining, Spatial Data Analysis, Text Similarity, and Uncertain Data Mining.

Zhihong Tian is currently a Professor, and Dean, with the Cyberspace Institute of Advanced Technology, Guangzhou University, Guangdong Province, China. Guangdong Province Universities and Colleges Pearl River Scholar (Distinguished Professor). He is also a part-time Professor at Carlton University, Ottawa, Canada. Previously, he served in different academic and administrative positions at the Harbin Institute of Technology. He has authored over 200 journal and conference papers in these areas. His research interests include computer networks and cyberspace security. His research has been supported in part by the National Natural Science Foundation of China, National Key Research and Development Plan of China, National High tech RD Program of China (863 Program), and National Basic Research Program of China (973 Program). He also served as a member, Chair, and General Chair of a number of international conferences. He is a Senior Member of the China Computer Federation, and a Senior Member of IEEE.