Efficient Size-Bounded Community Search over Large Networks

Kai Yao
The University of Sydney
Sydney, Australia
kyao8420@uni.sydney.edu.au

Lijun Chang
The University of Sydney
Sydney, Australia
Lijun.Chang@sydney.edu.au

ABSTRACT

The problem of community search, which aims to find a cohesive subgraph containing user-given query vertices, has been extensively studied recently. Most of the existing studies mainly focus on the cohesiveness of the returned community, while ignoring the size of the community, and may yield communities of very large sizes. However, many applications naturally require that the number of vertices/members in a community should fall within a certain range. In this paper, we design exact algorithms for the general size-bounded community search problem that aims to find a subgraph with the largest min-degree among all connected subgraphs that contain the query vertex q and have at least ℓ and at most h vertices, where q, ℓ, h are specified by the query. As the problem is NP-hard, we propose a branch-reduce-and-bound algorithm SC-BRB by developing nontrivial reducing techniques, upper bounding techniques, and branching techniques. Experiments on large real graphs show that SC-BRB on average increases the minimum degree of the community returned by the state-of-the-art heuristic algorithm GreedyF by a factor of 2.41 and increases the edge density by a factor of 2.2. In addition, SC-BRB is several orders of magnitude faster than a baseline approach, and all of our proposed techniques contribute to the efficiency of SC-BRB.

PVLDB Reference Format:
Kai Yao and Lijun Chang. Efficient Size-Bounded Community Search over Large Networks. PVLDB, 14(8): 1441 - 1453, 2021.
doi:10.14778/3457390.3457407

1 INTRODUCTION

Graph data is ubiquitous in real world applications, as the relationship among entities in the applications can be naturally captured by the graph model. Graph-based data analytics, as a result, has become increasingly popular, among which cohesive subgraph computation is a fundamental problem [9]. Many of the recent focus of cohesive subgraph computation is on cohesive subgraph search, also known as *community search*, which aims to find cohesive subgraphs that contain user-specified query vertices and possibly satisfy some other constraints [16, 22]. Community search has a wide range of applications, such as event organization [33, 34], social marketing [30], task scheduling [35], and social network analysis [24, 31].

The basic setting is as follows: given a large graph $G = (V, E)$ and a query vertex $q \in V$, community search aims to find a subgraph of G that contains q and is most cohesive (*e.g.*, densest). One of the most popular cohesiveness measures adopted in the literature is the *minimum degree* [6, 13, 36], which is also the one we adopt in this paper. That is, it aims to find the one with the largest min-degree among all subgraphs of G that contain q. As there could exist many subgraphs having the same min-degree, the existing studies return either a maximal one [6] or an arbitrary one [13, 36] among all subgraphs with the largest min-degree. The maximal subgraph with the largest min-degree can be computed in linear time [36], by iteratively peeling the vertex with the smallest degree; the optimal result is among these n (*i.e.*, $|V|$) subgraphs. However, the result could be extremely large which may overwhelm end-users [9].

It is observed in [36] that limiting the size of the returned community, to accommodate resource limitations, is natural and interesting from application point of view: for example, organize a hiking trip with up to 15 attendees, assemble a team of up to 10 workers for a project [29]. Motivated by this, the problem of community search with size restriction is formulated and studied in [36], which restricts the consideration to subgraphs with at most h vertices for a user-given h. The problem with size restriction becomes NP-hard, and two heuristic algorithms, GreedyD and GreedyF, are proposed in [36]. Both algorithms involve solving the problem without size restriction (*i.e.*, compute the maximal subgraph with the largest min-degree) on subgraphs of G. Specifically, GreedyD iteratively solves the problem without size restriction on the subgraphs of G induced by $V_{\leq d}$ for different d values, where $V_{\leq d}$ is the set of vertices that are within d hops from q in G. It returns the solution of $V_{\leq d^*}$ where d^* is the largest d such that the solution size of $V_{\leq d}$ is at most h; if there is no such d, then it returns the solution of $V_{\leq 1}$. For time efficiency consideration, GreedyF simply extracts the set $S \subseteq V$ of h vertices that are closest to q in G, and then returns the solution of the problem without size restriction on the subgraph of G induced by S.[1] Both algorithms have no guarantee on the minimum degree of the returned community compared with the optimal one (*i.e.*, largest min-degree). We observe in our experiments that the communities returned by GreedyD and GreedyF are far from being optimal. We also observe that GreedyD usually returns a community with larger min-degree than GreedyF, but the size of the community returned by GreedyD may exceed h.

In this paper, we aim to develop *exact* algorithms for the general *size-bounded community search* (SCS) problem, which in addition to the size upper bound h, also imposes a size lower bound ℓ. This is because some applications may also require the size of the identified community to be not too small. For example, consider the toy social network in Figure 1 of an event organization platform (*e.g.*, Meetup). Suppose user v_1 is planning to organize a group trip, and (s)he is deciding whom to invite. There are several considerations.

Proceedings of the VLDB Endowment, Vol. 14, No. 8 ISSN 2150-8097.
doi:10.14778/3457390.3457407

[1] Note that our descriptions here for GreedyD and GreedyF are simplified for the case that there is only one query vertex.

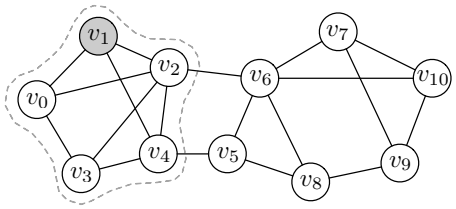


Figure 1: A toy graph

Firstly, to enjoy a friendly atmosphere during the trip, it is desirable that every attendee has as many friends in the group as possible. Secondly, due to the limitation of accommodation, there can be at most eight attendees. Thirdly, to be qualified for an air-ticket discount, the group should not be smaller than four. Thus, v_1 may issue an SCS query with size constraint $\ell = 4$ and $h = 8$, and find that $\{v_0, v_1, v_2, v_3, v_4\}$, which has minimum degree 3, is the best group. Note that, the existing approaches without size constraint may recommend the entire graph which also has minimum degree 3; however, this exceeds the accommodation capacity.

Formally speaking, given a large graph G, a query vertex q and a size constraint $[\ell, h]$, the SCS problem aims to find a subgraph with the largest min-degree among all connected subgraphs of G that contain q and have at least ℓ and at most h vertices. As expected, the SCS problem is NP-hard. A straightforward approach is to enumerate all subgraphs of G whose sizes are between ℓ and h and then identify the one with the largest min-degree. However, the time complexity would be $\Theta(n^h)$ which is too high to be practical. Note that, we cannot exploit the apriori-based pruning which has been demonstrated to be very successful for reducing the search space of frequent subgraph mining that also enumerates subgraphs of size up to a threshold [23]. This is because the property of minimum degree is not *hereditary*; that is, a graph can have subgraphs with smaller min-degree than itself, and may also have subgraphs with larger min-degree than itself.

In order to efficiently solve the SCS problem for large real graphs, we propose a branch-reduce-and-bound algorithm SC-BRB by developing nontrivial reducing techniques, upper bounding techniques, and branching techniques. Specifically, we develop three reduction rules, degree-based reduction, distance-based reduction, and inclusion-based reduction, to reduce the size of an instance before generating new branches/recursions. We also design three upper bounds, degree-based upper bound, neighbor reconstruction-based upper bound, and degree classification-based upper bound, for pruning branches. In addition, we also devise domination-based branching to reduce the number of branches. We rigorously proved the correctness of our techniques.

Contributions. Our main contributions are as follows.

- To the best of our knowledge, we are the first to study the general size-bounded community search (SCS) problem that has both a size lower bound and a size upper bound.
- We propose a branch-reduce-and-bound algorithm SC-BRB for solving the SCS problem exactly over large real graphs.
- We develop nontrivial reducing techniques, upper bounding techniques, and branching techniques for the SCS problem.

We conduct extensive experiments on large real graphs. It shows that compared with GreedyF, our algorithm SC-BRB on average increases the minimum degree of the returned community by a factor of 2.41 (and by absolute value 5.05) and increases the edge density (*i.e.*, $\frac{2|E|}{|V|(|V|-1)}$) by a factor of 2.2. In addition, SC-BRB is several orders of magnitude faster than a baseline approach, and all of our proposed techniques contribute to the efficiency of SC-BRB.

Related Works. The problem of community search aims to find a cohesive subgraph for user-given query vertices. The cohesiveness of a subgraph is usually measured, in the literature, by minimum degree [13, 26, 36], minimum number of triangles each edge participates in [21], or edge connectivity [8, 10]; see [9] for a survey. Most of the existing studies do not explicitly control the size of the returned community, and thus may output arbitrarily large communities. Sozio et al. [36] first studied the problem of size constrained community search (*i.e.*, SCS without size lower bound ℓ) in view of its importance, and proposed heuristic algorithms GreedyD and GreedyF. However, as observed in our experiments, the quality of the results returned by GreedyD and GreedyF are unsatisfactory. Approximation algorithms for the size constraint community search are investigated in [3]. Specifically, for any $\varepsilon > 0$, a solution with approximation ratio $2n^{\frac{1}{4}+\varepsilon}$ can be obtained in $n^{O(\frac{1}{\varepsilon})}$ time. In addition, a randomized $O(\sqrt{n \log n})$-approximation algorithm is proposed in [3]. However, these approximation algorithms are of theoretical interest only, and also size lower bound ℓ is not considered.

Li et al. [26] recently proposed a branch-and-bound algorithm PSA for the minimum k-core problem that aims to find the smallest subgraph with min-degree at least k and containing user-given query vertices. This problem is also NP-hard. In particular, PSA maintains a lower bound lb and an upper bound ub for the minimum k-core size, and terminates the algorithm once $\frac{ub}{lb} \leq c$ for a user-given approximation parameter $c \geq 1$. We show in our experiments that although PSA can be adapted to solve the special case of SCS problem without size lower bound (*i.e.*, $\ell = 1$), the performance is not satisfactory. Specifically, PSA does not return the optimal solution when $c > 1$, and runs extremely slow for exact computation (*i.e.*, $c = 1$). Moreover, PSA cannot be adapted to solve the general SCS problem with size lower bound. Note that, our techniques can potentially be used to speed up the computation of minimum k-core. That is, given an upper bound ub of the minimum k-core size, we will be searching for a subgraph of size at most $h = ub - 1$ and with minimum degree at least k; thus, our techniques can be applied.

Ma et al. [29] formulated the size-constrained k-core group query (SCGQ) over edge-weighted graphs, which aims to find a subgraph with the largest total weight (*i.e.*, number of edges for unweighted graphs) among all subgraphs that contain a query vertex q and exactly h vertices, and have minimum degree at least k. This problem is NP-hard, and an exact algorithm BS that traverses the solution space in breadth-first manner is proposed in [29]. We can adapt BS to solve our SCS problem by enumerating h and k. However, our experimental results show that BS is extremely inefficient due to lack of pruning and bounding techniques. Our techniques can also potentially be used to speed up SCGQ queries, the same as above.

Besides simple undirected graphs, community search has recently also been investigated for graphs associated with other information, *e.g.*, attributed graphs [20], geo-social graphs [11], temporal graphs [27], directed graphs [28], and heterogeneous graphs [17]; please refer to the two recent surveys [16, 22] for more details. Due

to inherently different graph models and moreover not considering size constraints, these techniques cannot be applied to solve our SCS problem. Another line of related works is geo-social group queries (GSGQ), which aims to find a fixed number of users who possess strong social connections with each other and have the minimum aggregated spatial distance to a rally point [19, 38, 39]. GSGQ is essentially different from our problem since it needs to consider the spatial location of each user and moreover it fixes the number of vertices/users to be a specific value. Thus, the proposed algorithms cannot be applied to our problem.

2 PRELIMINARIES

We focus on a large undirected and unweighted graph $G = (V, E)$, where V is the set of vertices and E is the set of edges. We denote the number of vertices and the number of edges in G by n and m, respectively. The set of neighbors of u in G is $N_G(u) = \{v \in V \mid (u, v) \in E\}$, and the degree of u in G is $d_G(u) = |N_G(u)|$. When the context is clear, $N_G(u)$ and $d_G(u)$ are simplified to $N(u)$ and $d(u)$, respectively. Given a vertex subset S of G, we use $G[S]$ to denote the subgraph of G induced by S, *i.e.*, $G[S] = (S, \{(u, v) \in E \mid u \in S, v \in S\})$. Given an arbitrary graph g, we use $V(g)$ and $E(g)$ to denote its set of vertices and its set of edges, respectively.

We measure the cohesiveness of a subgraph by its *minimum degree*. Minimum degree is a widely adopted cohesiveness measure in the literature [6, 13, 26, 36], due to its simplicity and easy computability. A related concept is *k-core*. Given a graph G and an integer k, the k-core of G is the *maximal* subgraph g of G such that $d_{\min}(g) \geq k$, where $d_{\min}(g)$ denotes the minimum degree of g. Note that, the k-core of G is unique for any $k \geq 0$, and is a vertex induced subgraph of G. The *core number* of a vertex v in G, denoted $\mathrm{cn}(v)$, is defined as the largest k such that the k-core of G contains v. It is well-known that computing the core number for all vertices in a graph, called the *core decomposition* problem, can be achieved in linear time [5, 32].

Problem Definition (Size-Bounded Community Search)**.** Given a graph $G = (V, E)$, a query vertex $q \in V$, and a size constraint $[\ell, h]$, the Size-Bounded Community Search (SCS) problem aims to find a subgraph H of G such that it satisfies all of the three conditions:

1) **Connected:** H is connected and contains q;
2) **Size Bounded:** H satisfies the size constraint, *i.e.*, $\ell \leq |V(H)| \leq h$;
3) **Cohesive:** The minimum degree of H is maximized among all subgraphs of G satisfying the above two conditions.

An SCS query consists of a query vertex q, size lower bound ℓ, and size upper bound h. Note that, the aim of SCS query is not to recover the ground-truth communities, but to find densely-connected subgraphs that are of a certain size (*i.e.*, between ℓ and h). The parameters ℓ and h should be specified based on the resource limitations of the applications as illustrated in Introduction. For presentation simplicity, we assume there is only a single query vertex; nevertheless, our techniques can be extended to handle multiple query vertices. We call a subgraph of G that satisfies the first two conditions (*i.e.*, connected, and size bounded) a *feasible community* (or feasible solution), and a subgraph satisfying all the three conditions an *optimal community* (or optimal solution). We call the minimum degree of an optimal community the *optimal min-degree*. Note that, the optimal community is not unique, but the optimal min-degree is unique. It is easy to see that if H is an optimal community to the SCS problem, then the subgraph of G induced by $V(H)$ is also an optimal community. Thus, *we consider only vertex-induced subgraphs in the remainder of the paper, and we denote a subgraph simply by its set of vertices.* The notations that are frequently used in the paper are summarized in Table 1.

Table 1: Frequently used notations

Notation	Definition
$N_S(u)$	the set of neighbors of u in the subgraph S
$d_S(v)$	the degree of vertex v in the subgraph S
$d_{\max}(S)$	maximum degree of the subgraph S
$d_{\min}(S)$	minimum degree of the subgraph S
$\mathrm{cn}(v)$	core number of vertex v in G
$dist_S(u, v)$	the shortest distance between u and v in the subgraph S
$C_{C,R}$	the set of all feasible communities X s.t. $C \subseteq X \subseteq C \cup R$
$\tilde{k}$ and $\hat{k}$	lower bound and upper bound of the optimal min-degree
$C^{\leq \tau}$	the set of vertices of degree $\leq \tau$, *i.e.*, $\{u \in C : d_C(u) \leq \tau\}$

Example 2.1. Consider the graph in Figure 1, and suppose the SCS query is $q = v_1$ and $[\ell, h] = [3, 5]$. Then, $H_1 = \{v_0, v_1, v_2\}$ is a feasible community which has minimum degree 2, while $H_2 = \{v_0, v_1, v_2, v_3, v_4\}$ is an optimal community which has minimum degree 3. □

We prove in the theorem below that the SCS problem is NP-hard.

Theorem 2.1. *The SCS problem is NP-hard.* [2]

Proof. We prove that the decision version of the SCS problem is NP-complete, by reducing from the k-clique problem which is NP-complete [18]. The decision SCS problem is as follows: given a graph $G = (V, E)$, a query vertex $q \in V$, a size constraint $[\ell, h]$ and an integer k, determine whether G has a subgraph g such that (1) g is connected and contains q, (2) $\ell \leq |V(g)| \leq h$, and (3) the minimum degree of g is at least k. It is obvious that the decision problem is in NP.

Now, given an instance of the k-clique problem which takes a graph $G = (V, E)$ and an integer k as input and aims to determine whether G has a k-clique (*i.e.*, a complete subgraph with k vertices), we reduce it into a decision SCS problem as follows. We add a dummy vertex v_q, and an edge between v_q and every vertex of V. Denote the resulting graph as G', *i.e.*, $V(G') = V \cup \{v_q\}$ and $E(G') = E \cup \{(v_q, v) \mid v \in V\}$. The query of our decision SCS problem consists of a query vertex v_q, size constraint $[k + 1, k + 1]$, and minimum-degree threshold k. It is easy to verify that $C \subseteq V$ is a k-clique if and only if $C \cup \{v_q\}$ is a solution to the decision SCS problem. Thus, the decision SCS problem is NP-complete, and the (optimization version of) SCS problem is NP-hard. □

3 A BASELINE APPROACH

In this section, we present a baseline approach for the SCS problem. It computes an optimal community by *enumerating* all feasible communities and identifying the one with the largest min-degree.

[2] Note that the proof in [36] reduces from the Steiner tree problem, and thus does not work for the special case that there is only one query vertex.

Algorithm 1: SC-Enum$(G, q, [\ell, h])$

```
1  Heuristically compute a feasible community H; /* e.g., invoke GreedyF */;
2  k̃ ← minimum degree of G[H];   /* Lower bound of optimal min-degree */;
3  Compute core number cn(·) for all vertices in G;
4  k̂ ← min{cn(q), h − 1};        /* Upper bound of optimal min-degree */;
5  if k̃ < k̂ then
6      Remove from G all vertices v satisfying cn(v) ≤ k̃;
7      Enum({v_q}, V(G) \ {v_q});
8  return H;

   Procedure Enum(C, R)
9  if |C| ∈ [ℓ, h] and d_min(C) > k̃ then
10     k̃ ← d_min(C); H ← C;
11 if |C| < h and R ≠ ∅ then
12     v ← a vertex from R;
13     Enum(C ∪ {v}, R \ {v});
14     Enum(C, R \ {v});
```

The pseudocode of the enumeration procedure Enum is shown in Lines 9–14 of Algorithm 1. Given a partial solution C and a candidate set R of vertices such that $C \cap R = \emptyset$, Enum aims to enumerate all feasible communities X such that $C \subseteq X \subseteq C \cup R$; denote the set of all such feasible communities by $\mathcal{C}_{C,R}$. Enum in addition maintains H, the currently found best community (*i.e.*, the one with the largest min-degree among all enumerated feasible communities), and $\tilde{k}$, the minimum degree of $G[H]$, which is a lower bound of the optimal min-degree. If C is a feasible community (*i.e.*, $|C| \in [\ell, h]$) and the minimum degree of $G[C]$ is larger than $\tilde{k}$ (Line 9), then Enum updates H and $\tilde{k}$ (Line 10); note that, C is assumed to always contain q, and for presentation simplicity, we also assume that C is connected. Then, to enumerate all feasible communities $\mathcal{C}_{C,R}$ for the instance (C, R), Enum partitions the enumeration space into $(C \cup \{v\}, R \setminus \{v\})$ and $(C, R \setminus \{v\})$ for an arbitrarily chosen vertex $v \in R$ and conducts a recursion on each of the two enumeration subspaces (Lines 12–14); note that, $\mathcal{C}_{C\cup\{v\},R\setminus\{v\}} \cap \mathcal{C}_{C,R\setminus\{v\}} = \emptyset$ and $\mathcal{C}_{C\cup\{v\},R\setminus\{v\}} \cup \mathcal{C}_{C,R\setminus\{v\}} = \mathcal{C}_{C,R}$. Moreover, as the feasible communities have a size upper bound h, we can stop the recursion if $|C| \geq h$ (Line 11). It can be verified by induction that invoking Enum with $C = \{q\}$ and $R = V(G) \setminus \{q\}$ successfully enumerates all feasible communities of G.

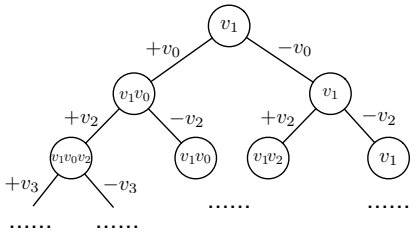


Figure 2: An example search tree

Example 3.1. For the graph and query in Example 2.1, Figure 2 illustrates a part of the search tree of the enumeration procedure Enum, where the root is the query vertex v_1. Initially, $C = \{v_1\}$ and $R = V(G) \setminus \{v_1\}$. In the first level, v_0 is selected from R and two branches/recursions are generated: the left branch moves v_0 from R to C, and the right branch discards v_0. For the first branch, we have $C = \{v_1, v_0\}$ and $R = V(G) \setminus \{v_1, v_0\}$, and we proceed forward by selecting the next unvisited vertex in R, *i.e.*, v_2. By moving v_2 from R to C, we get $C = \{v_1, v_0, v_2\}$, which is a feasible community with minimum degree 2. Thus we update $\tilde{k}$ as 2 and H as $\{v_1, v_0, v_2\}$. The search continues in a similar way until all the feasible communities are explored. Finally, we get the optimal solution $H = \{v_1, v_0, v_2, v_3, v_4\}$ with minimum degree 3. □

The time complexity of Enum is in the order of n^h, where n is the number of vertices in G and h is the size upper bound. To reduce n, we preprocess the graph G before invoking Enum. The pseudocode is given in Lines 1–8 of Algorithm 1. We first heuristically compute a feasible community H, *e.g.*, by invoking the heuristic algorithm GreedyF [36] (Line 1), and set the lower bound $\tilde{k}$ of the optimal min-degree as the minimum degree of $G[H]$ (Line 2). We also compute the upper bound $\hat{k}$ of the optimal min-degree as $\min\{\text{cn}(q), h-1\}$ (Line 3–4); it is easy to see that the optimal min-degree cannot be larger than $h-1$ and also cannot be larger than $\text{cn}(q)$. If $\tilde{k} = \hat{k}$, then H is already the optimal community. Otherwise, we shrink the graph G by removing all unpromising vertices (*i.e.*, the vertices v with $\text{cn}(v) \leq \tilde{k}$), and then invoke Enum on the reduced graph (Lines 5–6). The overall algorithm is denoted SC-Enum. Note that, to extend SC-Enum to handle multiple query vertices, we would need to (1) initially put all query vertices into the partial solution C at Line 7 and (2) check at Line 9 whether $G[C]$ is connected (if $G[C]$ is not connected, then we do not update $\tilde{k}$ and H at Line 10).

Limitations of the Baseline Approach. Despite the preprocessing of SC-Enum which reduces the input graph size, SC-Enum is still inefficient in processing large graphs. This is mainly due to the following three limitations of Enum.

(1) Large Candidate Set. Enum considers all vertices of R one by one, and in each recursion, it reduces the size of R only by one. Thus, the size of the candidate set R remains large during the recursion. However, given the current lower bound $\tilde{k}$ of the optimal min-degree, many vertices of R will not be part of any feasible community of $\mathcal{C}_{C,R}$ that has min-degree larger than $\tilde{k}$; thus, these vertices can be removed from R to reduce the search space.

(2) No Pruning based on Upper Bounds. For a given instance (C, R), Enum needs to enumerate all feasible communities of $\mathcal{C}_{C,R}$, *i.e.*, all subgraphs X such that $C \subseteq X \subseteq C \cup R$ and $\ell \leq |X| \leq h$. However, if we can compute an upper bound on the largest min-degree among all feasible communities in $\mathcal{C}_{C,R}$, then we can terminate/prune this instance if the upper bound is no larger than $\tilde{k}$. In this way, the search space will be significantly reduced.

(3) Naive Branching Rule. Enum arbitrarily selects one vertex $v \in R$ and then generates two recursions/branches, $(C \cup \{v\}, R \setminus \{v\})$ and $(C, R \setminus \{v\})$. This may generate a large number of recursions. Also, the ordering of generating the subinstances of Enum (*i.e.*, via selecting the vertex of R to branch) may affect the speed of tightening the lower bound $\tilde{k}$, and thus affect the search tree size.

4 A BRANCH-REDUCE-AND-BOUND APPROACH

In this section, we propose a branch-reduce-and-bound algorithm SC-BRB for solving the SCS problem. SC-BRB specifically addresses the three limitations of Enum by proposing branching techniques, reducing techniques, and upper bounding techniques. Note that designing branching, reducing, and bounding techniques is the standard approach (and usually the only approach) for exactly solving NP-hard problems [2, 37]. In the following, we present reducing techniques in Section 4.1, upper bounding techniques in Section 4.2, and branching techniques in Section 4.3. Finally, Section 4.4 presents the overall algorithm SC-BRB.

4.1 Reducing Techniques

Given an instance (C, R) of Enum, we propose reduction rules to reduce the size of R by either removing unpromising vertices from R or greedily moving promising vertices from R to C. Recall that, given an instance (C, R), we aim to find the feasible community in $C_{C,R}$ that has the largest min-degree if this min-degree is larger than $\tilde{k}$. That is, among all feasible communities in $C_{C,R}$, we are only interested in the ones that have min-degree larger than $\tilde{k}$, as the currently found best community has min-degree $\tilde{k}$.

The first reduction rule prunes a vertex v from R based on its degree in $C \cup R$ or its degree in $C \cup \{v\}$. All our proofs are omitted due to limit of space, and can be found in the full version [1].

Reduction Rule 1 (Degree-based Reduction)**.** Given an instance (C, R) and any vertex $v \in R$, if $\min\{d_{C\cup R}(v), d_{C\cup\{v\}}(v) + h - |C| - 1\} \leq \tilde{k}$, then we can discard v from R, where $d_{C\cup R}(v)$ is the degree of v in the subgraph $G[C \cup R]$.

The next reduction rule is based on a lower bound, as shown in the lemma below, on the size (*i.e.*, number of vertices) of a graph that has minimum degree k and diameter D. The *diameter* of a graph is the largest value among all pair-wise shortest distances, *i.e.*, $\max\{dist_G(u, v) \mid u, v \in V(G)\}$, where $dist_G(u, v)$ denotes the shortest distance between u and v in G.

Lemma 4.1. *Any graph of minimum degree $k \geq 1$ and diameter $D \geq 1$ must have at least $n(k, D)$ vertices, where*

$$n(k, D) = \begin{cases} k + D & \text{if } 1 \leq D \leq 2 \text{ or } k = 1 \quad (1a) \\ k + D + 1 + \lfloor \frac{D}{3} \rfloor (k - 2) & \text{otherwise} \quad (1b) \end{cases}$$

Moreover, this bound is tight; that is, for every $k \geq 1$ and every $D \geq 1$, there exists a graph with $n(k, D)$ vertices that has minimum degree k and diameter D.

Note that, the lower bound of Equation (1b) was stated in [15] for the case $D \geq 2$ and $k \geq 2$, but without detailed proof. We provide a detailed proof in the full version [1] for completeness. Our proof shows that the lower bound of Equation (1b) only holds for $D \geq 3$ and $k \geq 2$. Specifically, consider the graph that is a $(k + 2)$-vertex clique missing one edge which has minimum degree k and diameter $D = 2$; however, Equation (1b) would imply a lower bound of $k + 3$ for $D = 2$ which is incorrect. Thus, we provide a correct and tight lower bound of $k + D$ for the special case $1 \leq D \leq 2$ or $k = 1$.

Moreover, instead of using a closed formula to upper bound the diameter given h and k (*i.e.*, $\lfloor \frac{3h}{k+1} \rfloor - 1$ in [15]) which is not tight, we obtain the upper bound numerically. That is, the diameter is upper bounded by the largest D such that $n(k, D) \leq h$, which can be computed by binary search on D. This is because, $n(k, D)$ monotonically increases with both k and D. For example, for $h = 11$ and $k = 5$, the closed formula of [15] gives an upper bound of 4 for D, while our approach computes the upper bound as 2. Also note that, for a given size upper bound h, the maximum possible diameter D of a feasible community monotonically decreases when k increases. Computing the diameter, however, is computationally expensive. We use the following distance-based reduction, as the shortest distance between any pair of vertices is a lower bound of the diameter.

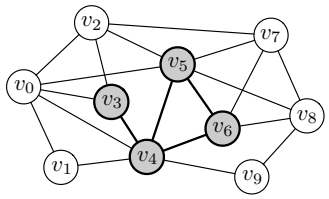


Figure 3: A running example for illustrating reduction rules

Reduction Rule 2 (Distance-based Reduction)**.** Given an instance (C, R) and any $v \in R$, if $n(\tilde{k} + 1, dist_{C\cup R}(u, v)) > h$, then we can discard v from R, where u is a vertex in C and $n(\cdot, \cdot)$ is the function defined in Lemma 4.1.

Besides discarding unpromising vertices from R, we can also greedily move promising vertices from R to C. The next reduction rule formulates such a condition.

Reduction Rule 3 (Inclusion-based Reduction)**.** Given an instance (C, R) and any $u \in C$, if $d_{C\cup R}(u) = \tilde{k}+1$, then we can greedily move to C all the vertices in R that are neighbors of u.

Example 4.1. Consider the graph in Figure 3, and suppose $[\ell, h] = [5, 7]$, $C = \{v_3, v_4, v_5, v_6\}$ (grey-shadowed nodes) and $R = \{v_0, v_1, v_2, v_7, v_8, v_9\}$ (white nodes), and $\tilde{k} = 2$. Recall that our aim is to find a feasible community in $C_{C,R}$ with minimum degree at least $\tilde{k}+1 = 3$. According to Reduction Rule 1, v_1 can be discarded because $d_{C\cup R}(v_1) = 2 \leq \tilde{k}$; similarly, v_9 can be discarded. From Lemma 4.1, we derive that the diameter of any feasible community with min-degree at least 3 is at most 2; this is because $n(3, 3) = 8 > h$. Thus, v_8 can be discarded by Reduction Rule 2, since $dist_G(v_3, v_8) = 3$. At last, as $d_{C\cup R}(v_3) = \tilde{k} + 1$, we move all neighbors of v_3 in R (*i.e.*, v_0 and v_2) to C. Similarly, we move the neighbors of v_6 in R (*i.e.*, v_7) to C. As a result, we obtain a feasible community with min-degree 3 by the reduction rules, *i.e.*, $\{v_0, v_2, v_3, v_4, v_5, v_6, v_7\}$. □

4.2 Upper Bounding Techniques

Let $\text{OptMD}(C, R)$ denote the largest min-degree among all feasible communities of $C_{C,R}$. In this subsection, we aim to compute upper bounds of $\text{OptMD}(C, R)$ such that the instance (C, R) can be entirely pruned if this upper bound is no larger than $\tilde{k}$. In the following, we first present the general idea of upper bound computation in Section 4.2.1, and then present three approaches to computing the upper bound in Sections 4.2.2–4.2.4. Finally, we put the three upper bounds together in Section 4.2.5.

4.2.1 General Idea of Upper Bounding. The reduction rules presented in Section 4.1 reduce the size of R by either discarding vertices from R or greedily moving vertices from R to C. This is achieved by inspecting individual vertices of C or R, and making local decisions based on properties of the individual vertices. Upper bounding $\text{OptMD}(C, R)$ in contrast aims to prune the entire instance (C, R), by considering all vertices of C and R altogether. Recall that computing $\text{OptMD}(C, R)$ is to identify the X such that

Condition (1) $C \subseteq X \subseteq C \cup R$,
Condition (2) $\ell \leq |X| \leq h$, and
Condition (3) $\min_{u \in X} d_X(u)$ is maximized.

Let X^* be the best community in $\mathcal{C}_{C,R}$, *i.e.*, satisfying the above three conditions. Note that, computing $\text{OptMD}(C, R)$ is NP-hard, as $\text{OptMD}(\{q\}, V(G) \setminus \{q\})$ is the optimal min-degree of the SCS problem. Thus, we compute an upper bound of $\text{OptMD}(C, R)$ by relaxing condition (2) and/or using upper bounds of $d_{X^*}(u)$. Specifically, we will compute an upper bound of $\min_{u \in C} d_{X^*}(u)$ by *only considering the degrees of vertices of* C, which obviously is an upper bound of $\min_{u \in X^*} d_{X^*}(u)$.

It is worth mentioning that, for our algorithm, it suffices to check whether the minimum degree of X^* is at least $\tilde{k}+1$. Nevertheless, we present our techniques for computing upper bounds, as computing upper bounds is more general in the sense that it could also be potentially used in other search paradigms (*e.g.*, best-first search). We will show in Section 4.2.5 that, with simple modifications, the upper bounding techniques can be used to efficiently check whether the minimum degree of X^* is at least $\tilde{k}+1$. Thus, we do not utilize $\tilde{k}$ in the following upper bounding techniques.

4.2.2 Degree-based Upper Bound. Firstly, we introduce the degree-based upper bound, denoted by U_d, by considering each vertex $u \in C$ individually and by upper bounding $d_{X^*}(u)$.

Lemma 4.2 (Degree-based Upper Bound)**.**

$$U_d = \min_{u \in C} \min\{d_{C \cup R}(u), d_C(u) + h - |C|\}$$

is an upper bound of $\text{OptMD}(C, R)$.

The degree-based upper bound is in analogous to the degree-based reduction rule presented in Section 4.1. But the degree-based upper bound considers degrees of vertices of C, while the degree-based reduction rule considers degrees of vertices of R.

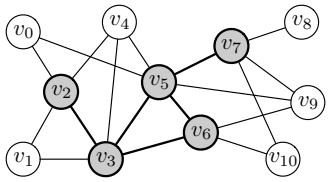


Figure 4: A running example for illustrating upper bounds

Example 4.2. Consider the graph in Figure 4, and suppose $[\ell, h] = [5, 8]$, $C = \{v_2, v_3, v_5, v_6, v_7\}$ and $R = \{v_0, v_1, v_4, v_8, v_9, v_{10}\}$. We use $\hat{d}(u)$ to denote $d_C(u) + \min\{h - |C|, d_{R \cup \{u\}}(u)\}$, where $h - |C| = 3$. The maximum possible degree of v_2 in X^* is 4, *i.e.*, $\hat{d}(v_2) = 1 + \min\{3, 3\} = 4$. For v_3, as it has only 2 neighbors outside C, *i.e.*, $d_{R \cup \{v_3\}}(v_3) = 2$, we have $\hat{d}(v_3) = 3 + \min\{3, 2\} = 5$. Similarly, we derive $\hat{d}(v_5) = 3 + \min\{3, 3\} = 6$, $\hat{d}(v_6) = 2 + \min\{3, 2\} = 4$ and $\hat{d}(v_7) = 1 + \min\{3, 3\} = 4$. Finally, we get $U_d = 4$. □

Time Complexity. In the implementation, we incrementally maintain the degrees $d_{C \cup R}(u)$ and $d_{C \cup \{u\}}(u)$ for each vertex $u \in C$. Thus, the time complexity of computing U_d is $O(|C|)$.

4.2.3 Neighbor Reconstruction-based Upper Bound. The degree-based upper bound U_d ignored the degrees of vertices of R and assumed that the degrees of vertices of $X^* \setminus C \subseteq R$ could be arbitrarily large. Here, we design the neighbor reconstruction-based upper bound, by explicitly considering the degrees of vertices of R. Specifically, we define the *neighbor reconstruction* problem: given a graph g, b vertices $v_1, \ldots, v_b \notin V(g)$, and b non-negative integers $d_1, \ldots, d_b$, how to add edges between $V(g)$ and $\{v_1, \ldots, v_b\}$ without parallel edges, such that degrees of $v_1, \ldots, v_b$ are $d_1, \ldots, d_b \in [0, |V(g)|]$, respectively, and the minimum degree among vertices of g in the resulting graph is maximized. Intuitively, by letting $g = G[C]$, $b = h - |C|$, $\{v_1, \ldots, v_b\}$ be the b vertices of R with the most number of neighbors in C, and d_i be the number of v_i's neighbors in C, then the minimum degree obtained by neighbor reconstruction is an upper bound of $\text{OptMD}(C, R)$.

We propose a greedy approach to solve the neighbor reconstruction problem. As the objective is to maximize the minimum degree among vertices of g in the reconstructed graph, v_i intuitively should be connected to vertices of g that have the smallest degrees. Thus, we process vertices $\{v_1, \ldots, v_b\}$ in an arbitrary order, and when processing v_i, we connect v_i to the d_i vertices in $V(g)$ that currently have the smallest degrees; note that, the degrees of $V(g)$ dynamically increase when new edges are added.

Algorithm 2: UB-NeighborReconstruct(C, R)

```
1 R' ← the h − |C| vertices in R that have the largest
    {d_{C∪{v}}(v) : v ∈ R};
2 for each u ∈ C do d_u ← d_C(u);
3 for each v ∈ R' do
4     C' ← the d_{C∪{v}}(v) vertices in C that have the smallest
        {d_u : u ∈ C};
5     for each u ∈ C' do d_u ← d_u + 1;
6 U_nr ← min_{u∈C} d_u;
7 return U_nr;
```

The pseudocode of neighbor reconstruction-based upper bound is shown in Algorithm 2, which is self-explanatory by following the above discussions.

Lemma 4.3 (Neighbor Reconstruction-based Upper Bound)**.** *Algorithm 2 returns a valid upper bound of* $\text{OptMD}(C, R)$.

Example 4.3. Consider the same setting as Example 4.2, Figure 5 illustrates the steps of computing U_{nr} by Algorithm 2. The initial degrees d_u of vertices of C are $\{v_2 : 1, v_3 : 3, v_5 : 3, v_6 : 2, v_7 : 1\}$ and the vertices of R with their numbers of neighbors to be reconstructed are $\{v_0 : 2, v_1 : 2, v_4 : 3, v_8 : 1, v_9 : 3, v_{10} : 2\}$. The first step is to select the $h - |C| = 3$ vertices of R that have the largest number of neighbors to be reconstructed, and $R' = \{v_4, v_9, v_0\}$. Then, we process the three vertices of R' sequentially. In the first

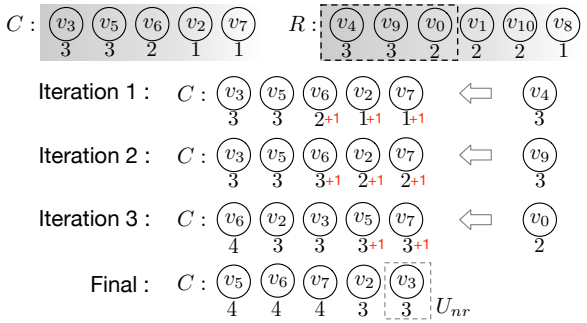


Figure 5: Neighbor reconstruction-based upper bound

iteration, we process v_4 by connecting it to the $d_{C\cup\{v_4\}}(v_4) = 3$ vertices of C that currently have the smallest degrees, and $C' = \{v_2, v_6, v_7\}$; after processing v_4, the degrees of vertices of C become $\{v_2 : 2, v_3 : 3, v_5 : 3, v_6 : 3, v_7 : 2\}$. In second iteration, we process v_9 similarly and the degrees of $C' = \{v_2, v_6, v_7\}$ increase by one each. In the third iteration, we process v_0 and the degrees of $C' = \{v_5, v_7\}$ increase by one each. Finally, the minimum degree of vertices of C is 3, which is returned as the upper bound U_{nr}. □

Time Complexity. Firstly, in our implementation of selecting R' at Line 1 of Algorithm 2, instead of considering all vertices of R which can be of large quantity, we only consider the vertices of R that is adjacent to some vertex of C. Thus, the time complexity of selecting R' is $O(d_{C,R})$ where $d_{C,R} = \sum_{u\in C} d_{R\cup\{u\}}(u)$; note that, selecting top-k numbers from a list of unsorted numbers can be achieved in linear time regardless of k [12]. Secondly, each vertex $v \in R'$ can be processed in $O(|C|)$ time. Thus, the total time complexity of Algorithm 2 is $O(d_{C,R} + (h - |C|)|C|)$.

4.2.4 Degree Classification-based Upper Bound. In the computation of U_{nr}, we utilized the count of all v_i's neighbors in C, $d_{C\cup\{v_i\}}(v_i)$, to decide the number of edges to be reconstructed between v_i and C. Let $C'' = \{u \in C : d_C(u) \geq U_{nr}\}$ be the set of vertices of C whose degrees in C are already no smaller than U_{nr}. Even if we remove all the edges between R and C'' before the neighbor reconstruction (equivalently, decrease the number of edges to be reconstructed between v_i and C), we intuitively will still get a valid (and likely smaller) upper bound of $\mathrm{OptMD}(C, R)$.

Formally, let $d_{\min}(C)$ and $d_{\max}(C)$ be the minimum degree and maximum degree of $G[C]$, respectively, and let $C^{\leq\tau}$ be the set of vertices of C whose degrees in C are at most τ, *i.e.*, $C^{\leq\tau} = \{u \in C : d_C(u) \leq \tau\}$. We prove a more general lemma in below.

Lemma 4.4. *For any $\tau \in [d_{\min}(C), d_{\max}(C)]$, let U^{τ}_{nr} be the result of running Algorithm 2 with the following modifications: remove all edges between R and $C \setminus C^{\leq\tau}$, and replace C with $C^{\leq\tau}$ at Lines 4 and 6. Then, U^{τ}_{nr} is a valid upper bound of* $\mathrm{OptMD}(C, R)$.

Following Lemma 4.4, we can compute an upper bound U^{τ}_{nr} for each $\tau \in [d_{\min}(C), d_{\max}(C)]$, and then return the minimum one $\min_{\tau\in[d_{\min}(C),d_{\max}(C)]} U^{\tau}_{nr}$ which is also a valid upper bound of $\mathrm{OptMD}(C, R)$. We call this upper bound, the degree classification-based upper bound. Its pseudocode is shown in Algorithm 3, where Lines 4–9 correspond to Lines 1–6 of Algorithm 2 with the modifications as described in Lemma 4.4; note that Line 11 should be ignored at the moment.

Algorithm 3: UB-DegreeClassification(C, R)

```
1  U_dc ← ∞;
2  for τ ← d_min(C) to d_max(C) do
3      C^{≤τ} ← {u ∈ C : d_C(u) ≤ τ};
4      R' ← the h − |C| vertices in R that have the largest
         {d_{C^{≤τ}∪{v}}(v) : v ∈ R};
5      for each u ∈ C^{≤τ} do d_u ← d_C(u);
6      for each v ∈ R' do
7          C' ← the d_{C^{≤τ}∪{v}}(v) vertices in C^{≤τ} that have the
             smallest {d_u : u ∈ C^{≤τ}};
8          for each u ∈ C' do d_u ← d_u + 1;
9      U^τ_nr ← min_{u∈C^{≤τ}} d_u;
10     U_dc ← min{U_dc, U^τ_nr};
11     if U_dc ≤ τ + 1 then break;
12 return U_dc;
```

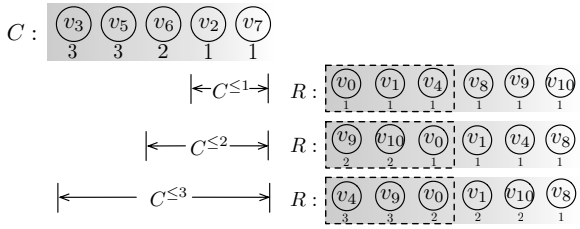


Figure 6: Degree classification-based upper bound

Corollary 4.5. *The degree classification-based upper bound is tighter than the neighbor reconstruction-based upper bound,* i.e., *$U_{dc} \leq U_{nr}$ for any instance (C, R).*

This directly follows from Lemma 4.4 and Algorithm 3, as $C^{\leq\tau} = C$ when $\tau = d_{\max}(C)$.

Example 4.4. Let's reconsider Example 4.3. We have $d_{\min}(C) = 1$, $d_{\max}(C) = 3$, $C^{\leq 1} = \{v_2, v_7\}$, $C^{\leq 2} = \{v_6, v_2, v_7\}$ and $C^{\leq 3} = \{v_3, v_5, v_6, v_2, v_7\}$, as shown in Figure 6. For each $\tau \in [1, 3]$, the numbers of edges to be reconstructed for vertices of R (*i.e.*, the number of neighbors in $C^{\leq\tau}$) are shown below the vertices in Figure 6. For $\tau = 1$, the top-3 vertices of R are $R' = \{v_0, v_1, v_4\}$. After the neighbor reconstruction operations as done by Lines 5–8 of Algorithm 3, d_{v_7} increases to 3 and d_{v_2} increases to 2, and thus $U^1_{nr} = 2$. Similarly, we will obtain $U^2_{nr} = 3$ and $U^3_{nr} = 3$; note that, the computation of U^3_{nr} is exactly the same as in Example 4.3, as $C^{\leq 3} = C$. Finally, we get $U_{dc} = \min\{2, 3, 3\} = 2$. □

Optimization. To compute U_{dc}, we may need to run neighbor reconstruction (*i.e.*, Lines 3–10 of Algorithm 3) for $d_{\max}(C)$ times, which in the worst case is $|C| - 1$. Thus, the computation of U_{dc} is costly. In the lemma below, we propose an early stop condition for computing U_{dc}, which corresponds to Line 11 of Algorithm 3.

Lemma 4.6. *For any $\tau \in [d_{\min}(C), d_{\max}(C)]$, if $\min^{\tau}_{i=d_{\min}(C)} U^i_{nr} \leq \tau + 1$, then $\min^{d_{\max}(C)}_{i=d_{\min}(C)} U^i_{nr} = \min^{\tau}_{i=d_{\min}(C)} U^i_{nr}$.*

Example 4.5. Let's continue Example 4.4. After computing $U^1_{nr} = 2$, it satisfies that $U^1_{nr} \leq 2 = \tau + 1$. Thus, we can terminate the computation and return $U_{dc} = 2$ directly, without computing U^2_{nr} nor U^3_{nr}. □

Time Complexity. As Algorithm 3 in the worst-case runs a modification of Algorithm 2 for $|C| - 1$ times, the time complexity of Algorithm 3 is $O\left(|C|\left(d_{C,R} + (h - |C|)|C|\right)\right)$. Note that, by the optimization at Line 11, the number of iterations could be much smaller than $|C| - 1$ in practice.

4.2.5 Putting the Upper Bounds Together. To maximally utilize the pruning power, we use all the three upper bounding techniques together and select the minimum one as UB, *i.e.*, $UB = \min\{U_d, U_{nr}, U_{dc}\}$. In the implementation, we first compute U_d, then U_{nr}, and finally U_{dc} in increasing order of their time complexities. Once a computed upper bound is enough to prune the instance (C, R), we terminate the upper bound computation immediately.

4.3 Branching Techniques

In this subsection, we propose two branching techniques, one for selecting which vertex to branch on, and another for how to generate the different branches.

Branching Vertex Selection. The lower bound $\tilde{k}$ of the optimal min-degree is critical to the search space size. Thus, we aim to identify feasible communities with large min-degree as early as possible. With this goal in mind, the vertex on which to branch intuitively should satisfy the following two conditions: (1) it should be connected to C; and (2) it should be adjacent to many low-degree vertices in C. To quantify this, we define the *connection score* for vertices of R.

Definition 4.1 (Connection Score)**.** Given an instance (C, R), the connection score of a vertex $v \in R$ is defined as

$$\delta(v) = \sum_{u \in N_{C \cup \{v\}}(v)} \frac{1}{d_C(u)}$$

Note that, if there is no edge between v and C, then $\delta(v) = 0$.

We choose the vertex of R that has the highest connection score, denoted v^*, to generate branches. Note that this naturally guarantees that C is always a connected subgraph during the recursions.

Domination-based Branching. After choosing v^* as described above, instead of generating two branches $(C \cup \{v^*\}, R \setminus \{v^*\})$ and $(C, R \setminus \{v^*\})$, we propose a domination-based strategy to reduce the number of generated branches.

Definition 4.2 (Vertex Domination)**.** Given an instance (C, R), vertex $v \in R$ *dominates* $v' \in R$, denoted $v \succeq v'$, if every neighbor of v' (in $C \cup R$) is either a neighbor of v or is v itself.

For instance, for the graph in Figure 4, v_4 dominates v_0 and v_1, and v_9 dominates v_8 and v_{10}.

Lemma 4.7. *Given an instance (C, R) and two vertices $v, v' \in R$, if v dominates v' (*i.e.*, $v \succeq v'$), then there is either a best community in $C_{C,R}$ (*i.e.*, with minimum degree* $\mathsf{OptMD}(C, R)$*) that contains both v and v', or a best community in $C_{C,R}$ that does not contain v'.*

Following Lemma 4.7, after choosing v^* that has the highest connection score, we generate branches as follows. If there is a vertex v' that is dominated by v^*, then we generate three branches $(C \cup \{v^*, v'\}, R \setminus \{v^*, v'\})$, $(C \cup \{v^*\}, R \setminus \{v^*, v'\})$ and $(C, R \setminus \{v^*, v'\})$, which reduces the number of branches from 4, as generated by the naive approach in Algorithm 1, to 3. Moreover, we generalize this idea to the case that v^* dominates multiple vertices. Let $\Phi = \{v_1, \ldots, v_l\}$ be the set of vertices of R that are dominated by v^*. Then, we generate the following branches:

- $(C \cup \{v^*, v_i\}, R \setminus \{v^*, v_1, \ldots, v_i\})$ for $1 \le i \le l$.
- $(C \cup \{v^*\}, R \setminus \{v^*, v_1, \ldots, v_l\})$.
- $(C, R \setminus \{v^*, v_1, \ldots, v_l\})$.

4.4 The SC-BRB Algorithm

Based on the techniques developed in the previous subsections, we propose the BRB algorithm for solving an instance (C, R). The pseudocode is shown in Algorithm 4, which is similar to Enum in Algorithm 1, but incorporates our newly proposed techniques. We first apply our reduction rules to reduce the size (C, R) (Line 1). If C is a feasible community and its min-degree is larger than $\tilde{k}$, then we update $\tilde{k}$ and H by C (Lines 2–3). Next, if the instance is not pruned by our upper bounding technique (Line 4), then we generate branches as follows. We select the vertex v^* from R that has the highest connection score (Line 5), identify the set $\Phi \subseteq R$ of vertices that are dominated by v^* (Line 6), and order the vertices in Φ in decreasing order based on their connections cores (Line 7). Finally, we generate $|\Phi| + 2$ branches/recursions.

Algorithm 4: BRB(C, R)

```
1  Reduce (C, R) by our reduction rules;
2  if ℓ ≤ |C| ≤ h and d_min(C) > k̃ then
3      k̃ ← d_min(C); H ← C;
4  if |C| < h and R ≠ ∅ and UB(C,R)> k̃ then
5      v* ← the vertex in R with the highest connection score;
6      Φ ← the vertices of R that are dominated by v*;
7      Order the vertices of Φ based on their connection scores, and
         let the ordered vertices be v_1, ..., v_l;
8      for i ← 1, ..., l do
9          BRB(C ∪ {v*, v_i}, R \ {v*, v_1, ..., v_i});
10     BRB(C ∪ {v*}, R \ {v*, v_1, ..., v_l});
11     BRB(C, R \ {v*, v_1, ..., v_l});
```

By replacing the invocation of Enum at Line 7 of Algorithm 1 with BRB, we have our branch-reduce-and-bound algorithm SC-BRB for the SCS problem.

5 A HEURISTIC APPROACH

In this section, we propose a heuristic algorithm SC-Heu for the SCS problem, which enforces that the size of the returned community is between ℓ and h. We replace GreedyF with SC-Heu in Algorithm 1 for efficiently computing an initial feasible community.

The pseudocode of SC-Heu is shown in Algorithm 5. We consider two cases. If the degree of q in G is no smaller than $h-1$ (Line 2), then we adopt the shrinking approach. It starts with the ego-network of q (Line 3) and then iteratively shrinks the graph by removing the vertex with the minimum degree (Line 7); the one with the largest min-degree among all generated feasible communities is returned as the result (Lines 5–6). Here, the ego-network of q in G is the subgraph of G induced by $\{q\} \cup N(q)$. If the degree of q in G is smaller than $h - 1$ (Line 8), then we use the expanding approach. It starts with the subgraph $\{q\}$ (Line 9) and then iteratively expands

Algorithm 5: SC-Heu(G, q, ℓ, h)

```
1  H ← ∅; k̃ ← 0;
2  if d_G(q) ≥ h − 1 then
3      S ← the ego-network of q;
4      while |S| ≥ ℓ do
5          if |S| ≤ h and d_min(S) > k̃ then
6              k̃ ← d_min(S); H ← S;
7          Delete from S the vertex with the minimum degree;
8  else
9      S ← {q};
10     while |S| < h do
11         v* ← the vertex with the highest connection score to S;
12         S ← S ∪ {v*};
13         if |S| ≥ ℓ and d_min(S) > k̃ then
14             k̃ ← d_min(S); H ← S;
15 return H;
```

the subgraph by including the vertex with the highest connection score into the subgraph (Line 11–12); the one with the largest min-degree among all generated feasible communities is returned as the result (Lines 13–14).

The time complexity of Algorithm 5 is $O(m + n \log n)$. Lines 2–7 run in $O(m)$ time similar to the peeling-based core decomposition algorithm [5]. Lines 9–14 run in $O(m + n \log n)$ time similar to Dijkstra's algorithm for computing single-source shortest paths [12].

6 EXPERIMENTS

In this section, we evaluate the effectiveness and efficiency of our techniques for the size-bounded community search problem.

Algorithms. We compare the following algorithms.

- GreedyF and GreedyD: the two heuristic algorithms proposed in [36]. Note that, it is straightforward to extend these two algorithms to consider the size lower bound ℓ.
- BS: the algorithm proposed in [29] for computing the size-constrained k-core over edge-weighted graphs. We adapt BS to our SCS problem by enumerating k and h.
- PSA: the progressive algorithm proposed in [26] for (approximately) computing a minimum k-core. We will describe how to adapt PSA in Section 6.3.
- SC-Enum: our baseline algorithm presented in Algorithm 1.
- SC-BRB: our algorithm proposed in Section 4.4.

All algorithms are implemented in C++ and run in main memory.

Datasets. We evaluate the algorithms on ten real graphs. UK2002 and Webbase are downloaded from WebGraph [7], while all the other graphs are downloaded from SNAP [25]. For each graph, we removed self-loops, parallel edges, as well as the direction of edges. Statistics of the graphs are shown in Table 2, where the graphs are listed in increasing order regarding their numbers of edges; $k_{\max}$ is the maximum k such that the graph contains a non-empty k-core.

Besides real graphs, we also generate synthetic graphs to evaluate the efficiency of the algorithms. Specifically, we generated four power-law graphs PL1, PL2, PL3, PL4 by GTgraph [4], and four graphs SBM1, SBM2, SBM3, SBM4 by the stochastic block model [14]. All the synthetic graphs have 10^6 vertices, and we vary the number of edges from 10^6 (*i.e.*, PL1 and SBM1) to 10^9 (*i.e.*, PL4 and SBM4) with an increasing factor of 10. For the stochastic block model graphs, the number of communities is set as 10^4, and thus each community contains 100 vertices.

Table 2: Statistics of real datasets

Dataset	n	m	d_{avg}	$d_{\max}$	$k_{\max}$
Email	36, 692	183, 831	10.02	1, 383	43
HepPh	34, 546	420, 877	24.36	846	30
DBLP	317, 080	1, 049, 866	6.62	343	113
YouTube	1, 134, 890	2, 987, 624	5.26	28, 754	51
Google	875, 713	4, 322, 051	9.87	6, 332	44
BerkStan	685, 230	6, 649, 470	19.40	84, 230	201
Gplus	107, 614	12, 238, 285	227.44	20, 127	752
Flickr	1, 715, 254	15, 551, 249	18.13	27, 224	568
UK2002	18, 459, 128	261, 556, 721	28.33	194, 955	943
Webbase	118, 142, 155	1, 019, 903, 190	17.26	816, 127	1, 506

Parameters and Query Generation. We vary the size upper bound h from 6 to 18 with increment 3. For each h, we set the size lower bound ℓ to be $h - 3$. That is, the size constraint $[\ell, h]$ is selected from $\{[3, 6], [6, 9], [9, 12], [12, 15], [15, 18]\}$. For each size constraint $[\ell, h]$, we randomly generate 100 queries, where the query vertex of each query is randomly selected from the set of vertices with core number larger than 5; this is similar to previous studies [17] and is to ensure that there is a meaningful community containing the query vertex. Besides these queries, we also fix one of ℓ and h and vary the other (see Sections 6.2 and 6.3 for details), and we randomly select query vertices from different parts of the graph (i.e., Dense and Sparse queries in Section 6.2).

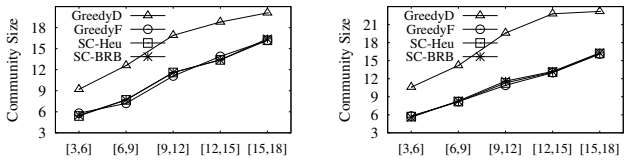
For each $[\ell, h]$, the average result quality and processing time of the 100 queries are reported. For each testing, we set a time limit of two hours, and if an algorithm does not finish within the time limit, we record its running time as inf. All experiments are conducted on a machine with an Intel Core-i7 3.20GHz CPU and Ubuntu system.

6.1 Effectiveness of Our Algorithms

In this subsection, we evaluate the effectiveness of our exact algorithm SC-BRB against the existing heuristic algorithms GreedyF and GreedyD. We exclude BS, as it is an exact algorithm and thus the minimum degrees of its reported communities are the same as that of SC-BRB. We also compared the minimum degree, edge density, and overlap ratio between the ground truth communities and that returned by our algorithm SC-BRB, despite that the goal of SC-BRB is not to recover the ground truth communities; the results are reported in the full version [1].

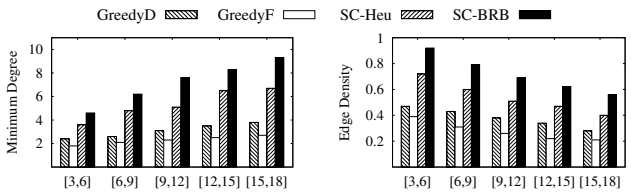
Result Size. Figure 7 reports the sizes of the communities returned by GreedyF, GreedyD, SC-Heu and SC-BRB, where the size constraint $[\ell, h]$ varies from $[3, 6]$ to $[15, 18]$. We can see that the result sizes of GreedyF and both of our algorithms fall within the range $[\ell, h]$, but GreedyD reports communities with more than h vertices. This is because GreedyF and our algorithms deliberately enforce the size constraint, while GreedyD does not enforce that.

Result Quality. To evaluate the result quality, we report the minimum degree and edge density (*i.e.*, $\frac{2m}{n(n-1)}$) of the extracted communities. For both metrics, the larger the value, the better the quality.



(a) Email (b) DBLP

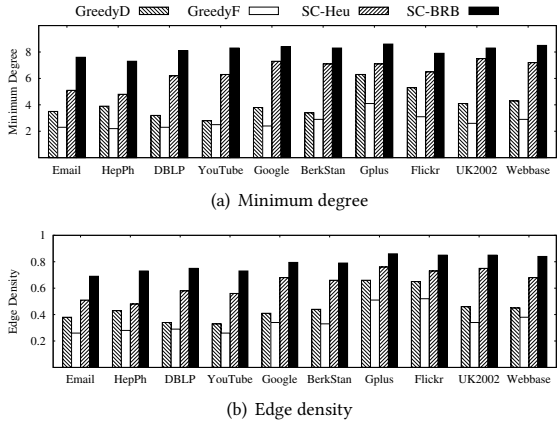
Figure 7: Result size



(a) Minimum degree (Email) (b) Edge density (Email)

(c) Minimum degree (DBLP) (d) Edge density (DBLP)

Figure 8: Result quality by varying $[\ell, h]$

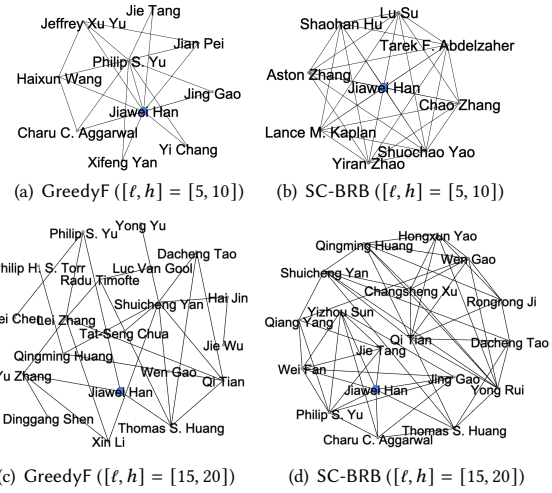


(a) Minimum degree

(b) Edge density

Figure 9: Result quality on all graphs ($[\ell, h] = [9, 12]$)

The results on Email and DBLP by varying $[\ell, h]$ are shown in Figure 8. We can see that GreedyD has a slightly higher result quality than GreedyF, but at the cost of violating the size constraint. Nevertheless, both of our algorithms significantly improve the result quality compared with GreedyD, and our exact algorithm SC-BRB has the highest result quality. The results on all the graphs by fixing $[\ell, h] = [9, 12]$ are shown in Figure 9, which have similar trends as in Figure 8. Compared with GreedyF, SC-BRB on average increases the minimum degree by a factor of 2.41 (by absolute value 5.05), and improves the edge density by a factor of 2.2. The average degrees of the communities computed by these algorithms are reported in the full version [1]. The results are similar to that for minimum degree and edge density.



(a) GreedyF ($[\ell, h] = [5, 10]$) (b) SC-BRB ($[\ell, h] = [5, 10]$)

(c) GreedyF ($[\ell, h] = [15, 20]$) (d) SC-BRB ($[\ell, h] = [15, 20]$)

Figure 10: Case studies

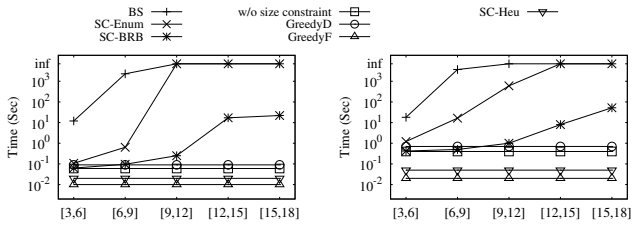
Case Study. We construct another coauthor graph, denoted DBLP_{cs}, from the raw DBLP dataset [3] for case study. Each vertex corresponds to an author, and two vertices are connected by an edge if the two authors have coauthored at least five papers. The DBLP_{cs} graph contains 424,784 vertices and 888,392 edges. In the case study, we search for the size-bounded community for "Jiawei Han", a renowned researcher in Data Mining. Firstly, the results returned by GreedyF and SC-BRB for size constraint $[5, 10]$ are shown in Figure 10(a) and 10(b). We can see that the result by GreedyF is sparse and has minimum degree 2 and edge density 0.48, while the result by our algorithm SC-BRB has minimum degree 7 and edge density 0.91. Figure 10(c) and 10(d) illustrate the results for size constraint $[15, 20]$. The result by GreedyF has minimum degree 1 and edge density 0.15, while the result by SC-BRB has minimum degree 5 and edge density 0.37.

Suppose Jiawei Han would like to assemble a team to identify and tackle a grand problem in data mining. Then he can issue an SCS query with ℓ and h being specified based on the team size. If the ideal team size is between 5 and 10, then the best team could be the one in Figure 10(b) as each member has collaborated with at least 7 other members in the team. If the ideal team size is between 15 and 20, then the best team could be the one in Figure 10(d); note that, with the size constraint $[15, 20]$, there is no team such that every member has collaborated with ≥ 6 or ≥ 7 other members.

6.2 Efficiency Testings

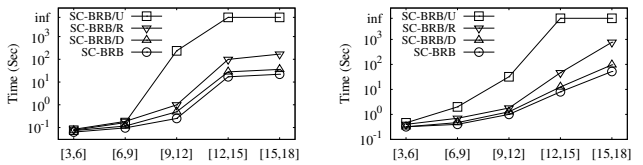
Against Baseline Algorithms. We first evaluate SC-BRB against baseline algorithms SC-Enum and BS. We adapted BS proposed in [29] to solve the SCS problem by enumerating k and $x \in [\ell, h]$ to find the largest k such that there is a k-core of size x. The running time of these algorithms on DBLP and Google by varying $[\ell, h]$ is shown in Figure 11. We can see that SC-BRB significantly outperforms SC-Enum and BS due to our powerful pruning and bounding techniques. Moreover, BS due to lack of pruning and bounding techniques is also outperformed by SC-Enum. Thus, we exclude SC-Enum and BS from the remaining testings.

[3] https://dblp.uni-trier.de/xml/



(a) DBLP (b) Google

Figure 11: Against baseline algorithms (vary $[\ell, h]$)



(a) DBLP (b) Google

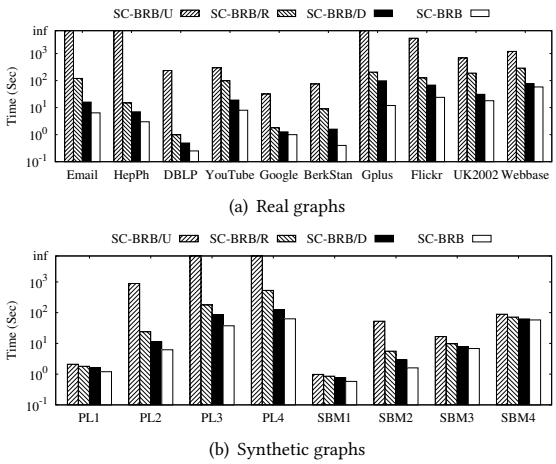
Figure 12: Evaluate our different techniques (vary $[\ell, h]$)

In Figure 11, we also include the running time of SC-Heu, GreedyD, GreedyF, and community search w/o size constraint. These heuristic algorithms run much faster than the exact algorithms. However, as demonstrated in Section 6.1, the result quality of GreedyD and GreedyF are not satisfactory. On the other hand, community search w/o size constraint will return extremely large communities (*e.g.*, almost the entire graph) [9]. Consider the significant improvements on the result quality, it is cost-effective to apply SC-BRB. On the other hand, if time efficiency is critical, then our heuristic algorithm SC-Heu is recommended as it is superior than other alternatives.

Reducing, Upper Bounding, and Branching Technique. To evaluate our different techniques, we also implemented SC-BRB/R which is SC-BRB without our reduction rules, SC-BRB/U which is SC-BRB without our upper bounding techniques, and SC-BRB/D which is SC-BRB without our domination-based branching. Note that, SC-BRB/D still uses our connection score-based branching vertex selection. The running time of these algorithms on DBLP and Google by varying $[\ell, h]$ is shown in Figure 12. We can see that the running time of all algorithms increases when the size constraint increases, this is because the search space becomes larger. Nevertheless, SC-BRB consistently outperforms all other algorithms. We can also see that the running time increases, whenever any of the reducing, upper bounding, and branching techniques is removed. This confirms that all our techniques make a contribution to the performance of SC-BRB.

The running time of the algorithms on all real and synthetic graphs for $[\ell, h] = [9, 12]$ is shown in Figure 13. The trends are similar to Figure 12. For synthetic graphs, we also observe that the running time increases when the number of edges (correspondingly, density) of the graph increases. Nevertheless, the increase is sublinear; recall that the number of edges of PL4 (resp. SBM4) is 10^3 times that of PL1 (resp. SBM1).

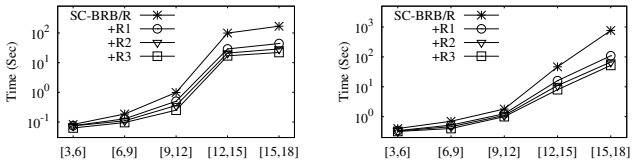
Evaluate Different Reduction Rules. We evaluate the effectiveness of the different reduction rules in reducing the running time. For simplicity, we use R1, R2 and R3 to represent our three reduction rules. We incrementally integrate these reduction rules into



(a) Real graphs

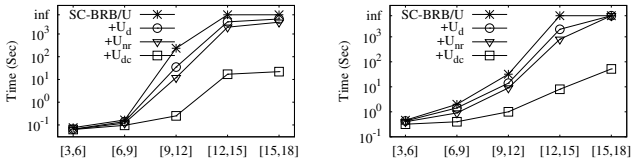
(b) Synthetic graphs

Figure 13: Running time on all graphs ($[\ell, h] = [9, 12]$)



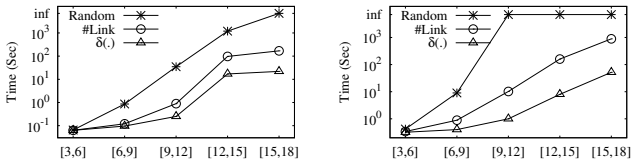
(a) DBLP (b) Google

Figure 14: Evaluate different reduction rules



(a) DBLP (b) Google

Figure 15: Evaluate different upper bounds



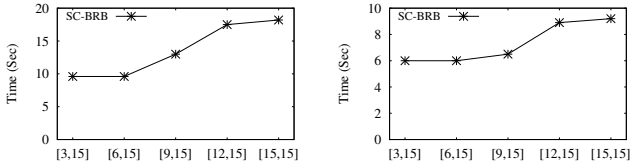
(a) DBLP (b) Google

Figure 16: Evaluate different branching vertex selection

SC-BRB/R. The results in Figure 14 show that each of the three reduction rules reduces the running time.

Evaluate Different Upper Bounds. Figure 15 demonstrates the effectiveness of the three upper bounds, U_d, U_{nr} and U_{dc}, by incrementally adding them to SC-BRB/U. Each of the upper bounds reduces the running time. Thanks to the degree classification strategy, U_{dc} produces a tighter upper bound and is most powerful.

Evaluate Different Branching Vertex Selection. Figure 16 evaluates the effectiveness of the branching vertex selection. All our algorithms adopt *connection score*-based vertex selection, denoted $\delta(\cdot)$. Besides $\delta(\cdot)$, we also implement two other strategies: Random, and #Link, which selects the vertex of R that has the most number of



(a) DBLP (b) Google

Figure 17: Evaluate different query-range sizes

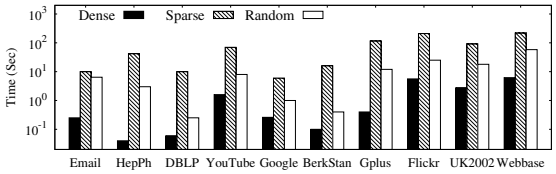


Figure 18: Different query types ($[\ell, h] = [9, 12]$)

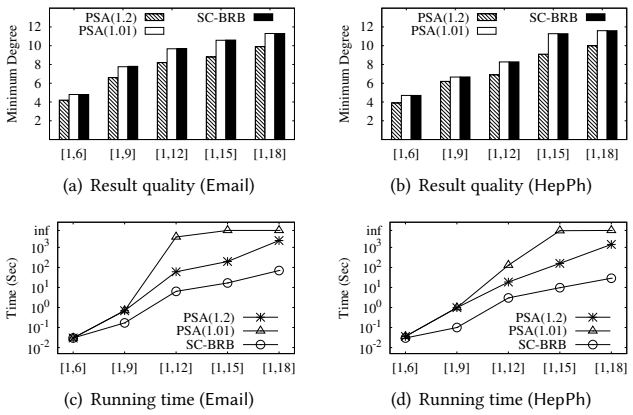
links to C. The results demonstrate that the connection score-based strategy improves the efficiency significantly.

Evaluate Different Query-Range Sizes. We now evaluate the performance of SC-BRB under different query-range sizes. We fix $h = 15$ and increase ℓ from 3 to 15. Figure 17 shows that the running time increases along with the increasing of ℓ. This is because, when ℓ increases, it becomes harder to find a high-quality solution at early stages. Nevertheless, the influence of ℓ on the running time is not as strong as that of h (*e.g.*, as shown in Figure 11), as the running time of SC-BRB is upper bounded by n^h.

Evaluate Different Query Types. We evaluate the efficiency of SC-BRB for different query types: Dense, Sparse and Random. Random queries are generated as described at the beginning of this section. Dense query vertices are uniformly selected from the last 10% of the degeneracy ordering of all vertices (*i.e.*, from dense parts of the graph), while Sparse query vertices are uniformly selected from the first 10% of the degeneracy ordering with core number at least 5 (*i.e.*, from sparse parts of the graph). The running time on all graphs for $[\ell, h] = [9, 12]$ is shown in Figure 18. We can see that Dense queries are the easiest to process, because we are more likely to find a high quality heuristic solution for Dense queries. Thus, our reduction rules and upper bounding techniques can significantly reduce the search space. Due to opposite reasons, Sparse queries are the hardest to process. Random queries are in between because they contain both Dense and Sparse queries.

6.3 SCS without Size Lower Bound

In this subsection, we evaluate SC-BRB against PSA for the problem of SCS without size lower bound. PSA was originally proposed for (approximately) computing the minimum k-core for a user-given integer k and query vertex q [26], *i.e.*, the smallest one among all subgraphs that contain q and have minimum degree at least k. In addition, PSA accepts a parameter $c \geq 1$ to strike a balance between the result quality and running time. Specifically, PSA returns a subgraph that is at most $c\times$ (the optimal size), instead of the optimal one. Intuitively, the smaller the value of c, the longer the running time. We adapt PSA for SCS without size lower bound, by computing an approximate minimum k-core for an iteratively increasing k until the size of the identified subgraph is larger than h. We report the



(a) Result quality (Email) (b) Result quality (HepPh)

(c) Running time (Email) (d) Running time (HepPh)

Figure 19: Comparison with PSA by varying $[\ell, h]$

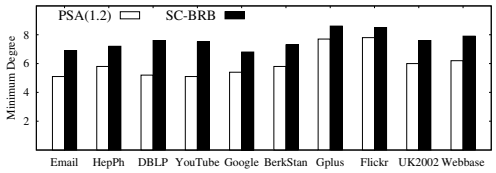


Figure 20: Comparison with PSA on all graphs ($h = 10$)

penultimate k value as the result quality of PSA. The source code of PSA is obtained from the authors of [26].

The running time and result quality of PSA(1.01), PSA(1.2) and SC-BRB on Email and HepPh by varying h are shown in Figure 19, where PSA(1.01) and PSA(1.2), respectively, represent PSA with $c = 1.01$ and $c = 1.2$. We can see that PSA(1.01) has the same result quality as SC-BRB due to the small value of $c = 1.01$, but is much slower than PSA(1.2) and SC-BRB and thus is not practical for large graphs. SC-BRB outperforms PSA(1.2) in terms of both result quality and running time.

Figure 20 shows the minimum degrees of the communities returned by PSA(1.2) and SC-BRB on all the graphs for $h = 10$. We can see that SC-BRB consistently outperforms PSA(1.2).

7 CONCLUSION

In this paper, we studied the problem of size-bounded community search which has both size lower bound and size upper bound, and aims to maximize the minimum degree of the returned subgraph. We proposed a branch-reduce-and-bound algorithm SC-BRB for solving the problem over large real graphs. SC-BRB outputs the optimal results. The efficiency of SC-BRB is due to our newly developed reducing techniques, upper bound techniques, and branching techniques. Extensive experiments on large real graphs demonstrate that SC-BRB significantly improves the result quality over the existing algorithms, and SC-BRB can efficiently process large graphs thanks to our advanced techniques. As a possible direction of future work, it will be interesting to adapt our techniques to speed up the computation for the problems studied in [26] and [29].

ACKNOWLEDGMENTS

This work was supported by the Australian Research Council Funding of FT180100256.

REFERENCES

[1] [n.d.]. full version: https://lijunchang.github.io/pdf/scs-tr.pdf.

[2] Faisal Abu-Khzam, Sebastian Lamm, Matthias Mnich, Alexander Noe, Christian Schulz, and Darren Strash. 2020. Recent Advances in Practical Data Reduction. *CoRR* abs/2012.12594 (2020).

[3] Omid Amini, David Peleg, Stéphane Pérennes, Ignasi Sau, and Saket Saurabh. 2012. On the approximability of some degree-constrained subgraph problems. *Discret. Appl. Math.* 160, 12 (2012), 1661–1679.

[4] David A Bader and Kamesh Madduri. 2006. Gtgraph: A synthetic graph generator suite. *Atlanta, GA, February* 38 (2006).

[5] Vladimir Batagelj and Matjaz Zaversnik. 2003. An O (m) algorithm for cores decomposition of networks. *arXiv preprint cs/0310049* (2003).

[6] Fei Bi, Lijun Chang, Xuemin Lin, and Wenjie Zhang. 2018. An Optimal and Progressive Approach to Online Search of Top-K Influential Communities. *Proc. VLDB Endow.* 11, 9 (2018), 1056–1068.

[7] Paolo Boldi and Sebastiano Vigna. 2004. The webgraph framework I: compression techniques. In *Proceedings of the 13th international conference on World Wide Web*. 595–602.

[8] Lijun Chang, Xuemin Lin, Lu Qin, Jeffrey Xu Yu, and Wenjie Zhang. 2015. Index-based optimal algorithms for computing steiner components with maximum connectivity. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. 459–474.

[9] Lijun Chang and Lu Qin. 2018. *Cohesive Subgraph Computation over Large Sparse Graphs*. Springer Series in the Data Sciences.

[10] Lijun Chang, Jeffrey Xu Yu, Lu Qin, Xuemin Lin, Chengfei Liu, and Weifa Liang. 2013. Efficiently computing k-edge connected components via graph decomposition. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2013*. 205–216.

[11] Lu Chen, Chengfei Liu, Rui Zhou, Jianxin Li, Xiaochun Yang, and Bin Wang. 2018. Maximum co-located community search in large scale social networks. *Proceedings of the VLDB Endowment* 11, 10 (2018), 1233–1246.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, 3rd Edition*. MIT Press.

[13] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*. 991–1002.

[14] Aurelien Decelle, Florent Krzakala, Cristopher Moore, and Lenka Zdeborová. 2011. Asymptotic analysis of the stochastic block model for modular networks and its algorithmic applications. *Physical Review E* 84, 6 (2011), 066106.

[15] Paul Erdös, János Pach, Richard Pollack, and Zsolt Tuza. 1989. Radius, diameter, and minimum degree. *J. Comb. Theory, Ser. B* 47, 1 (1989), 73–79.

[16] Yixiang Fang, Xin Huang, Lu Qin, Ying Zhang, Wenjie Zhang, Reynold Cheng, and Xuemin Lin. 2020. A survey of community search over big graphs. *The VLDB Journal* 29, 1 (2020), 353–392.

[17] Yixiang Fang, Yixing Yang, Wenjie Zhang, Xuemin Lin, and Xin Cao. 2020. Effective and efficient community search over large heterogeneous information networks. *Proceedings of the VLDB Endowment* 13, 6 (2020), 854–867.

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

[19] Bishwamittra Ghosh, Mohammed Eunus Ali, Farhana M Choudhury, Sajid Hasan Apon, Timos Sellis, and Jianxin Li. 2018. The flexible socio spatial group queries. *Proceedings of the VLDB Endowment* 12, 2 (2018), 99–111.

[20] Xin Huang and Laks VS Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the VLDB Endowment* 10, 9 (2017), 949–960.

[21] Xin Huang, Laks VS Lakshmanan, Jeffrey Xu Yu, and Hong Cheng. 2015. Approximate closest community search in networks. *Proceedings of the VLDB Endowment* 9, 4 (2015), 276–287.

[22] Xin Huang, Laks V. S. Lakshmanan, and Jianliang Xu. 2019. *Community Search over Big Graphs*. Morgan & Claypool Publishers.

[23] Akihiro Inokuchi, Takashi Washio, and Hiroshi Motoda. 2000. An Apriori-Based Algorithm for Mining Frequent Substructures from Graph Data. In *Principles of Data Mining and Knowledge Discovery, 4th European Conference, PKDD 2000, Lyon, France, September 13-16, 2000, Proceedings*, Djamel A. Zighed, Henryk Jan Komorowski, and Jan M. Zytkow (Eds.), Vol. 1910. 13–23.

[24] David Knoke and Song Yang. 2019. *Social network analysis*. Vol. 154. Sage Publications.

[25] Jure Leskovec and Andrej Krevl. 2014. SNAP Datasets: Stanford large network dataset collection.

[26] Conggai Li, Fan Zhang, Ying Zhang, Lu Qin, Wenjie Zhang, and Xuemin Lin. 2019. Efficient progressive minimum k-core search. *Proceedings of the VLDB Endowment* 13, 3 (2019), 362–375.

[27] Rong-Hua Li, Jiao Su, Lu Qin, Jeffrey Xu Yu, and Qiangqiang Dai. 2018. Persistent community search in temporal networks. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. IEEE, 797–808.

[28] Qing Liu, Minjun Zhao, Xin Huang, Jianliang Xu, and Yunjun Gao. 2020. Truss-based community search over large directed graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 2183–2197.

[29] Yu-Liang Ma, Ye Yuan, Fei-Da Zhu, Guo-Ren Wang, Jing Xiao, and Jian-Zong Wang. 2019. Who should be invited to my party: A size-constrained k-core problem in social networks. *Journal of Computer Science and Technology* 34, 1 (2019), 170–184.

[30] Puneet Manchanda, Grant Packard, and Adithya Pattabhiramaiah. 2015. Social dollars: The economic impact of customer participation in a firm-sponsored online customer community. *Marketing Science* 34, 3 (2015), 367–387.

[31] John Scott. 1988. Social network analysis. *Sociology* 22, 1 (1988), 109–127.

[32] Stephen B Seidman. 1983. Network structure and minimum degree. *Social networks* 5, 3 (1983), 269–287.

[33] Jieying She, Yongxin Tong, Lei Chen, and Caleb Chen Cao. 2016. Conflict-aware event-participant arrangement and its variant for online setting. *IEEE Transactions on Knowledge and Data Engineering* 28, 9 (2016), 2281–2295.

[34] Jieying She, Yongxin Tong, Lei Chen, and Tianshu Song. 2017. Feedback-aware social event-participant arrangement. In *Proceedings of the 2017 ACM International Conference on Management of Data*. 851–865.

[35] Oliver Sinnen and Leonel A Sousa. 2005. Communication contention in task scheduling. *IEEE Transactions on parallel and distributed systems* 16, 6 (2005), 503–515.

[36] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. 939–948.

[37] Gerhard J. Woeginger. 2001. Exact Algorithms for NP-Hard Problems: A Survey. In *5th International Workshop of Combinatorial Optimization - Eureka, You Shrink!, Papers Dedicated to Jack Edmonds*, Vol. 2570. 185–208.

[38] De-Nian Yang, Chih-Ya Shen, Wang-Chien Lee, and Ming-Syan Chen. 2012. On socio-spatial group query for location-based social networks. In *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*. 949–957.

[39] Qijun Zhu, Haibo Hu, Cheng Xu, Jianliang Xu, and Wang-Chien Lee. 2017. Geo-social group queries with minimum acquaintance constraints. *The VLDB Journal* 26, 5 (2017), 709–727.