Local Search of Communities in Large Graphs

Wanyun Cui[§] Yanghua Xiao[§*] Haixun Wang[‡†] Wei Wang[§]
[§]Shanghai Key Laboratory of Data Science, Fudan University, Shanghai, China
[‡]Microsoft Research Asia, Beijing, China
wanyuncui1@gmail.com, shawyh@fudan.edu.cn, haixun@gmail.com, weiwang1@fudan.edu.cn

ABSTRACT

Community search is important in social network analysis. For a given vertex in a graph, the goal is to find the best community the vertex belongs to. Intuitively, the best community for a given vertex should be in the vicinity of the vertex. However, existing solutions use *global search* to find the best community. These algorithms, although straight-forward, are very costly, as all vertices in the graph may need to be visited. In this paper, we propose a *local search* strategy, which searches in the neighborhood of a vertex to find the best community for the vertex. We show that, because the minimum degree measure used to evaluate the goodness of a community is not *monotonic*, designing efficient local search solutions is a very challenging task. We present theories and algorithms of local search to address this challenge. The efficiency of our local search strategy is verified by extensive experiments on both synthetic networks and a variety of real networks with millions of nodes.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Data mining*

Keywords

Community search; Social networks; Graph mining

1. INTRODUCTION

Most real-life complex networks, including the Internet, social networks, and biological neural networks, contain *community structures*. That is, the networks can be partitioned into groups within which connections are dense and between which connections are sparse [1]. Finding communities in real networks is an important analytical task, because community structures are imbued with meaning – that is, they are highly correlated with the functionality of the network. For example, on the World Wide Web, communities consist of web sites that share common topics [2]. In protein-protein interaction networks [3] and metabolic networks [4], community structures correspond to functionality modules.

Due to the significance of community structures, the problem of *community search*, that is, finding the most likely community that a vertex belongs to, is important to many real life networks and applications [5]. Typical applications include:

- *Friend recommendation on Facebook.* Given the friendship network, the system wants to suggest candidate friends to a specific user u. Intuitively, we will only recommend to u those who are in the same community as u are but are not yet u's friends.
- *Advertising on social networks.* People in the same community often share common interests. If we know a user is interested in a particular type of advertisements, we may push the same type of advertisements to other people in his or her community.
- *Infectious disease control.* If a person has come into contact with a highly infectious disease, then who will most likely be affected? Obviously, we may want to monitor people in his or her community.
- *Semantic expansion.* In information retrieval, when a user submits a keyword query, say, "image," he may also be interested in results related to other keywords such as "pictures," "photo," and so on. If a *semantic link network* [6] over keywords is available, then we can expand the query by including keywords in the same "semantic community."

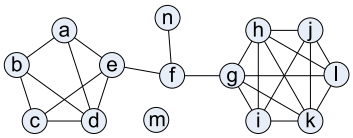


Figure 1: An example graph

To effectively perform community search, we must first determine what a good community is. The goodness of a community is measured by the *closeness* of the vertices in the community. One widely used closeness measure is a community's *minimum degree*, that is, the minimum degree of all the vertices in the subgraph induced by the community. A good community is one that has a large minimum degree. We illustrate this in Example 1.

EXAMPLE 1 (MINIMUM DEGREE). *Suppose we want to find the best community that contains vertex a in Figure 1. Intuitively, $V_1 = \{a, b, c, d, e\}$ forms a community that is highly connected with a. In fact, the subgraph induced by V_1 has a minimal degree of 3. If we include one more vertex f in the community, the minimal degree will drop to 1.*

Another possible measure is *average degree*. Under this measure, the best community for a will also include vertex f and a dense subgraph $V_2 = \{g, h, i, j, k, l\}$, as the average degree of $V_1 \cup \{f\} \cup V_2$ is approximately 3.8, which is larger than the average degree of V_1 (which is 3.2). Intuitively, however, V_1 and V_2

*Correspondence author. This work was supported by the National NSFC (No. 61003001, 61170006, 61171132, 61033010); NSF of Jiangsu Province (No. BK2010280).

†Current affiliation: Google Research, Mountain View, CA.

SIGMOD'14, June 22–27, 2014, Snowbird, UT, USA.

http://dx.doi.org/10.1145/2588555.2612179 .

are more likely to be two separate communities, as they are connected only by a weak link through f. Hence, the minimum degree measure captures the intuitive notion of closeness better. In this paper, we adopt the minimum degree measure for community search. Thus, the community search problem will be finding the connected subgraph with the largest minimal degree for a given query vertex.

Besides the above rationale, the following reasons also motivate us to take the minimum degree as the measure of community goodness. First, minimum degree is one of the most fundamental characteristics of a graph. For example, it is used to describe the evolution of random graphs [7], and graph visualization [8]. The properties found in this paper about minimal degree can shed light on other related problems. Second, in social network analysis, minimal degree has been widely used as the measure of the cohesiveness of a group of persons [9, 5, 10]. It can be dated back to the Seidman's research in 1983 [9]. The author compared minimal degree to many other measures of cohesiveness, such as connectedness and diameter, finding that minimum degree is a good measure in social network analysis. In a recent community search study [5], the authors also used the minimum degree as the community measure for organizing a cock tail party. They also found that, in community search problem, minimum degree is better than some other measures, such as average degree and density (measured by $\frac{2|E|}{|V|(|V|-1)}$, where $|V|$ and $|E|$ are the vertex number and edge number, respectively).

A straightforward strategy for community search is known as the *global search* [5]. The search is pessimistic in the sense that it needs to explore the entire graph before deciding the best community for the query vertex. Note that in the worst case, the whole network may be the best community. For example, when the whole network forms a complete graph.

In Example 2, we demonstrate a typical global search procedure on the graph shown in Figure 1. It starts with the whole graph and then it iteratively removes vertices that cannot be part of the answers. The procedure is repeated until no vertex can be removed.

EXAMPLE 2 (GLOBAL SEARCH). *Suppose we want to find the best community for vertex j in the network shown in Figure 1. We repeatedly remove vertices with the minimum degree. Vertices m, n, f, a, b, c, d, e could be removed in turn, and we end up with $V' = \{g, h, i, j, k\}$, wherein j is one of the vertices with the minimum degree. We can show that any subset of V' has a smaller minimal degree. Hence, V' is the best answer.*

Global search is costly as it needs to explore the entire graph. This is unacceptable for large graphs. In this paper, we propose a *local search* strategy. The intuition is that the best community for a given vertex is in the neighborhood of the vertex. Thus, it should not be necessary to involve the entire graph in the search. The local search strategy works as follows: We start at the query vertex. Initially the target community contains the query vertex only. The community expands as we explore in the neighborhood of the query vertex. We stop the search when the current community is the best.

The problem is, how do we know whether the current community we have found is the best? It would be nice if the minimum degree measure is "monotonic," meaning as the community becomes larger, its minimum degree always becomes smaller. Then, we can stop the search when the minimum degree drops below the given threshold. Unfortunately, the minimum degree measure is not monotonic. This is demonstrated by Example 3.

EXAMPLE 3 (LOCAL SEARCH AND NON-MONOTONICITY). *Suppose we want to find the best community for vertex a in Figure 1. Assume the current community includes a's immediate neighbors: b, d, e. The subgraph induced by $S = \{a, b, d, e\}$ has a minimal degree of 2. To enlarge the community, we can add vertex c or f. Adding f will decrease the minimal degree to 1, while adding c will increase it to 3.*

Clearly, as shown by Example 3, the minimum degree measure is not *monotonic*, which means that if S is a partial result containing the query vertex, for a vertex v that is adjacent to S, the subgraph induced by $S \cup \{v\}$ does not necessarily have a smaller or larger minimum degree. This poses a great challenge for us to decide when to terminate the local search procedure.

In this paper, we present theories and algorithms for local search. We investigate sufficient conditions for deciding whether a neighboring vertex should be added to expand the community. If no vertex satisfies the condition, then we terminate the search. We show that such a sufficient condition does exist, but, in the worst case, its evaluation may become as costly as performing a global search. However, a *typical* local search always finds the best community with much less cost than a global search.

The rest of the paper is organized as follows. Section 2 introduces some background information and formulates two community search problems: CST and CSM. Section 3 discusses how to solve the problems using global community search. In Section 4, we give several local search based solutions for the community search with a threshold constraint (CST) problem. In Section 5, we present solutions to the community search with a maximality constraint (CSM) problem. Section 6 reports experimental results. Section 7 reviews related work. We conclude in Section 8.

2. PROBLEM DEFINITION

In this section, we define a "goodness" measure for communities and we define the problems of community search. We then discuss the hardness of the problems. In this work, we are concerned with *simple graphs*, that is, graphs with no self-loops or multi-edges. Also, we focus on *undirected* and *unweighted* graphs only.

$G(V, E)$	A graph with vertex set V and edge set E.
$G[H]$	The subgraph of G induced by a set of vertex H. It contains edges $(H \times H) \cap E$.
$deg_G(v)$	The degree of vertex v in G.
$\delta(G)$	$min\{deg_G(v) \vert v \in V\}$
$H^*(G, v)$	Any best communities for vertex v in G.
$m^*(G, v)$	The community goodness of $H^*(G, v)$.
$V_{\geq k}$	$\{v \vert deg_G(v) \geq k\}$
$C_{naive}(k)$	Naive set of candidates for $\texttt{CST}(k)$.
$maxcore(G, v_0)$	The maximum core of G w.r.t v_0

Table 1: Notations

2.1 Problem Definition

Let $G(V, E)$ denote an undirected graph with vertex set V and edge set E. For any subset $H \subseteq V$, the *subgraph induced by* S, denoted as $G[H]$, is the graph whose vertex set is H and whose edge set is $(H \times H) \cap E$. Furthermore, we use $deg_G(v)$ to denote the degree of vertex v in graph G. Clearly, we have $deg_{G[H]}(v) \leq deg_G(v)$ since $G[H]$ is a subgraph of G. Table 1 summarizes the notations used in this paper.

DEFINITION 1 (COMMUNITY GOODNESS [5]). *Let $G(V, E)$ be a graph. Let $H \subseteq V$. We consider $G[H]$, the subgraph induced by H, as a community. The community goodness of $G[H]$ is defined by its minimum degree:*

$$\delta(G[H]) = min\{deg_{G[H]}(v) \vert v \in H\}$$

Minimum degree is one of the most widely used community goodness measures [5]. One important characteristics of $\delta(\cdot)$, which we have illustrated in Example 3, is that it is *non-monotonic*. It means $\delta(G[H \cup \{v\}])$ is not necessarily smaller than $\delta(G[H])$. Because of the non-monotonicity of $\delta(\cdot)$, the problem of finding the best community through local search is nontrivial.

PROBLEM DEFINITION 1 (CSM). *For a graph $G(V, E)$ and an arbitrary vertex $v_0 \in V$, find $H \subseteq V$ such that (1) $v_0 \in H$; (2) $G[H]$ is a connected subgraph; and (3) $\delta(G[H])$ is maximized among all possible choices of H. We denote this problem as CSM, or community search with the maximality constraint.*

For graph G and query vertex v, let $m^*(G, v)$ denote the possible maximum community goodness, and let $H^*(G, v)$ denote any community that has the maximum goodness. We have:

$$0 \leq m^*(G, v) \leq deg_G(v) \quad (1)$$

Note that the optimal solution is not necessarily unique. In general, $m^*(G, v)$ is determined by v and the graph structure of G.

In some applications, instead of finding communities that have the maximum community goodness, we may be interested in finding those that satisfy $\delta(G[H]) \geq k$, where k is a given constraint. Consider the *infectious disease control* problem. Infectious diseases have different risk factors. For a highly contagious disease, we may want to choose a lower k so that people who have little contact with the sick can also be found. For a less contagious disease, we may choose a higher k to focus on people who have very close contact with the sick. By tuning k, we achieve flexibility in controlling the size of the resulting community.

PROBLEM DEFINITION 2 (CST). *For a given graph $G(V, E)$, a query vertex $v_0 \in V$, and a constant k, find $H \subseteq V$ such that (1) $v_0 \in H$; (2) $G[H]$ is a connected graph; and (3) $\delta(G[H]) \geq k$. We denote this problem as $\texttt{CST}(k)$, or community search with threshold constraint.*

EXAMPLE 4. *We use the graph shown in Figure 1 as an example. Suppose the query vertex is a. In CSM, the subgraph induced by $H = \{a, b, c, d, e\}$ is the final solution because $\delta(G[H]) = 3$ and other Hs will have $\delta(G[H]) < 3$. In $\texttt{CST}(k)$, if $k = 3$, then the solution is still H. If $k = 2$, then there are multiple choices for H. For example, $\{a, b, d\}$, $\{a, d, e\}$ or $\{a, b, c, d, e\}$.*

We illustrate CSM and CST in Example 4. Both CSM and CST may return multiple answers. In fact, as shown in Section 2.2, CST may produce an exponential number of results (with regard to graph size). Hence, we only focus on finding one solution for CST. Alternatively, we may want to find the smallest community. This new problem, which is denoted as mCST, is formally below:

PROBLEM DEFINITION 3 (MCST). *For a graph $G(V, E)$, a query vertex $v_0 \in V$ and a constant k, find $H \subseteq V$ such that (1) $v_0 \in H$; (2) $G[H]$ is a connected graph; (3) $\delta(G[H]) \geq k$; and (4) the size of H is minimized. We denote this problem as $\texttt{mCST}(k)$.*

Unfortunately, the mCST problem is NP-complete as we will prove in Section 2.3. In this paper we focus on CST and CSM.

2.2 Relationship between CSM and CST

CSM is an optimization problem, and it has a corresponding decision problem: deciding whether a subset $H \subseteq V$ exists that satisfies the three conditions specified in CST. Clearly, CST is the *constructive version* of the decision problem of CSM. In other words, we not only need to determine the existence of the valid solution but also need to construct an instance of the solution if the valid solution does exist. Besides these obvious relationships, we further establish some quantitative correlations between CSM and CST that will serve as the foundation for solving these problems.

PROPOSITION 1 (DOWNWARD CLOSURENESS OF $\texttt{CST}(k)$). *If H is a solution to $\texttt{CST}(k)$, then H is also a solution to $\texttt{CST}(k')$ where $k' < k$.*

PROPOSITION 2. *For a graph $G(V, E)$ and a query vertex v, if H is a solution of $\texttt{CST}(k)$, then $m^*(G, v)$ is no less than k.*

PROPOSITION 3 (PRUNING RULE). *For a vertex v with $deg_G(v) < k$, v will not belong to any solutions to $\texttt{CST}(k)$.*

Proposition 1 and 2 lead to an algorithm of polynomial complexity to solve CSM if we have a polynomial solution for CST. Since $m^*(G, v)$ lies in the interval $[0, deg_G(v)]$, we can employ a binary search procedure starting from $\lceil \frac{N-1}{2} \rceil$ to iteratively visit the median of the interval to check whether the CST instance with the median as the threshold constraint has a valid solution. In this manner, we can solve CSM in $O(\log deg_G(v) f(N))$, where $f(N)$ is the time complexity of CST.

For both CSM and CST, we only look for one solution because there may exist an exponential number of solutions. To see this, consider a graph with N vertices of degree 1 and one vertex of degree N (Figure 2). Suppose the query vertex v_c is the vertex of degree N. Then, obviously we have $m^*(G, v_c) = 1$. But $H^*(G, v_c)$ could be any subset containing v_c. Thus the number of optimal solutions is $\Theta(2^n)$, where n is the number of all vertices of degree 1. To avoid returning an exponential number of solutions, in our problem, we only consider one of them.

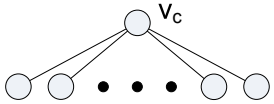


Figure 2: A graph with 1 vertex of degree N and N vertices of degree 1.

2.3 NP-completeness of mCST

Now we show that the mCST problem, which we introduced in Section 2.1, is NP-complete. To do this, we prove a problem related to mCST is NP-Complete, and we reduce that problem to mCST.

LEMMA 1. *For a given graph G, a vertex v_0, and an integer k, if there exists a clique C such that $v_0 \in C$ and $|C| = k + 1$, then C is the smallest solution of $\texttt{CST}(k)$.*

PROBLEM DEFINITION 4 (MCC). *For a graph $G(V, E)$ and $v_0 \in V$, find the maximum clique that contains v_0.*

LEMMA 2. *MCC is NP-Complete.*

PROOF. It is clear that MCC belongs to NP. We prove it by reducing Maximal Clique (MC), a well-known NP-complete problem, to MCC. For any graph $G(V, E)$, we construct a new graph $G'(V', E')$ by adding a vertex v_0 and connecting v_0 to all vertices in G, that is, $V' = V \cup \{v_0\}$ and $E' = E \cup \{(v_i, v_0) | v_i \in V\}$. Thus, an MCC in G' is an MC in G. □

THEOREM 1. *mCST is NP-complete.*

PROOF. It is easy to see that mCST belongs to NP. Now we show that mCST is NP-complete by reducing MCC to it. Let $G(V, E)$ be a graph, and let v_0 be the query vertex. Consider the decision problem that corresponds to the optimization problem of MCC: Determine whether G has a clique of at least size k that contains v_0. We can construct the following decision problem for mCST. Determine whether a solution $H \subseteq V$ exists such that $|H| = k$ and it satisfies the mCST conditions: i) $v_0 \in H$; ii) $G[H]$ is connected; and iii) $\delta(G[H]) \geq k - 1$. If H is the answer, then apparently $G[H]$ is a clique, as any node in H has degree $\geq k - 1$. □

3. GLOBAL SEARCH

In this section, we describe global search based solutions for CST and CSM. The global search approach needs to visit all vertices and edges of the graph, which is costly for big graphs.

3.1 k-core and Maxcore

To understand global search, we first define two concepts: *k-core* and *maxcore*.

DEFINITION 2 (k-CORE). *A subgraph of G is called the k-core of G if it is the biggest subgraph such that each of its vertices has a degree of at least k. Note that the k-core may contain multiple connected components.*

DEFINITION 3 (MAXIMUM CORE). *For a vertex v, a* maximum core with regard to v, *denoted as $maxcore(v)$, is the k-core with the maximum k among all the k-cores that contain v.*

EXAMPLE 5. *Consider graph G in Figure 1. The subgraph induced by $\{a, b, c, d, e, g, h, i, j, k, l\}$ is the 3-core of G; the subgraph induced by $\{g, h, i, j, k, l\}$ is the 4-core of G, which is also the maximum core of G; and the subgraph induced by $\{a, b, c, d, e\}$ is a maximum core with regard to vertex e, that is, $maxcore(G, e)$.*

Example 5 illustrates the concepts of k-core and $maxcore$. We use global search to find the k-core for a given k: We repeatedly remove vertices from G whose degree is less than k until no more vertices can be removed. We need to visit every vertex to find its degree and explore all of the edges in the graph. Hence, the complexity for finding the k-core and $maxcore$ is $O(|V| + |E|)$.

3.2 Solving CST and CSM

For a graph G and a query vertex v_0, we now show that the k-core and the $maxcore(v_0)$ contain the solutions to CST and CSM, respectively. First, consider the problem of CST(k). If vertex v belongs to any solution of CST(k), we call v an *admissible vertex*. Let A be the set of all admissible vertices for CST(k). Similarly, we can define the admissible set A' for the problem of CSM. Example 6 illustrates the admissible sets in Figure 1:

EXAMPLE 6. *Consider graph G in Figure 1 and query vertex e. For CSM, we have $m^*(G, e) = 3$ and $H^*(G, e) = \{a, b, c, d, e\}$. Since no other $H^*(G, e)$ exists, the admissible set is simply $A = \{a, b, c, d, e\}$. For CST(2), the solutions include $\{a, b, c, d, e\}$ and $V - \{m, n\}$. Hence, we have $A = V - \{m, n\}$.*

Next, we show that A is a subset of the k-core of G. Similarly, A' is contained in $maxcore(v_0)$. More specifically, we have:

LEMMA 3. *For graph G and query vertex v_0, the connected component C_k that contains v_0 in the k-core of graph G is a solution of CST(k). Furthermore, for any other solution H of CST(k), we have $H \subset C_k$.*

LEMMA 4. *For graph G and query vertex v_0, the connected component that contains v_0 in $maxcore(v_0)$, denoted by $C_{max}(v_0)$, is a solution of CSM. Furthermore, for any other solution H of CSM, we have $H \subset C_{max}(v_0)$.*

PROOF. By definition, $C_{max}(v_0)$ is a solution of CSM. If there exists another solution $H \neq C_{max}(v_0)$, then $H \cup C_{max}(v_0)$ will be a connected component in $maxcore(v_0)$, as both H and $C_{max}(v_0)$ contain v_0. This contradicts the definition of $C_{max}(v_0)$. □

Lemma 3 implies that a vertex v is an admissible vertex of CST(k) if and only if $v \in C_k$; and Lemma 4 implies that v is an admissible vertex of CSM if and only if $v \in C_{max}$. Unfortunately, evaluating these sufficient and necessary conditions is equivalent to global search.

To solve CST, Lemma 3 implies that we can iteratively remove vertices of less than k degree and their incident edges. Then, the connected component that contains the query vertex is certainly a valid solution.

To solve CSM, Lemma 4 implies that we need to find the connected component containing the query vertex in the maximum core with regard to the query vertex. We do this by using a greedy algorithm [5]. Let $G_0 = G$. We delete from G_0 the vertex with the minimum degree as well as its incident edges, and we denote the resulting graph as G_1. We repeat the process and create a sequence of graphs, $G_0, G_1, \cdots, G_t$, until at step t the query vertex v_0 is the next vertex to be deleted. Then, the connected component of G_i $(0 \leq i \leq t)$ with maximum $\delta(G_i)$ that contains the query vertex is an optimum solution [5].

The above two solutions have time complexity $O(|V| + |E|)$[1], which means we need to visit all vertices and edges in the graph.

[1]The linear complexity of a global search based CSM solution is achieved with some special technique. Please refer to [5] for its linear implementation.

4. LOCAL SEARCH FOR CST

In this section, we devise local search algorithms for community search. The biggest challenge is to overcome the non-monotonicity of the minimum degree measure, which enables us to perform community search by exploring only the local neighborhood of the query vertex. In the following, we first present a baseline local search solution, which is of exponential complexity. Then, we present the general framework of our linear solution. Finally, we give optimized realizations of this framework in Section 4.3.1 and Section 4.3.2. As we mentioned, CSM can be solved based on the solutions to the corresponding CST problem. Hence, we focus on CST only.

4.1 The Baseline Solution

We first give an in-depth analysis of the monotonicity of $\delta(\cdot)$, the community goodness function and introduce some notations. Consider the exploration starting from the query vertex. At each step we add a vertex until we get a solution H. Let $v_0, v_1, ..., v_t$ be a sequence of vertices which lead to H (we refer to such a sequence as *a sequence of H*). Let $H_i = \{v_0, \cdots, v_i\}$. We have shown that in general $\delta(\cdot)$ is a non-monotonic function of H. More formally, $\delta(H_i)$ is not necessarily smaller than $\delta(H_{i+1})$. Clearly, the monotonicity of $\delta(H)$ depends on the order in which vertices are added to H. One interesting fact we discovered is that for any vertex $v_0 \in H$, we can always find a vertex sequence (each vertex is in H) starting with v_0 such that $\delta(H_i)$ is a *non-decreasing* function of i.

THEOREM 2. *For any vertex $v_0 \in H$ in graph G, there always exists a vertex sequence $v_0, v_1, ..., v_t$ of H starting with v_0 such that $\forall 0 \leq i < t$, $\delta(H_i) \leq \delta(H_{i+1})$.*

PROOF. This is equivalent to proving that we can remove vertices one by one from H until v_0 such that the removal of each vertex will not increase the minimal degree of the remaining vertices. Suppose the current set is H'. If $H' = \{v_0\}$, we have already found the vertex sequence. If $H' = \{v_0, v_i\}$, then removing v_i either decreases the minimal degree (if $(v_0, v_i) \in E$) or does not change the minimal degree (if $(v_0, v_i) \notin E$). Next, we consider the case in which there are two or more vertices besides v_0 in H'. In such a case, there must be a vertex $v \in H', v \neq v_0$ and $\delta(G[H']) \geq \delta(G[H' - \{v\}])$. We just need to remove this vertex. If such vertices don't exist, it means that $\forall v \in H', v \neq v_0, \delta(G[H']) < \delta(G[H' - \{v\}])$. This only happens when v is one of the vertices in H' that has the minimal degree because removing a non-minimal degree vertex will only keep the minimal degree or decrease the minimal degree. Thus, there must be two or more such v. Removing each one of these vertices will not produce a larger minimal degree. □

Theorem 2 implies that there is always an order of exploration that monotonically leads to a solution. Theorem 2 also implies any solution H of CST(k) can be produced by a vertex sequence of H starting from the query vertex v_0 such that $\delta(H_i) \leq \delta(H_{i+1})$ for each i. In general the existence of such a sequence is a necessary but not a sufficient condition to find a valid solution. To see the insufficiency, consider the graph shown in Figure 1 and CST(3) with query vertex e. Any vertex sequence starting with e, f will not lead to a valid solution, but clearly $\delta(G[e, f])$ is larger than $\delta(G[e])$.

Theorem 2 leads to a straightforward algorithm which is outlined in Algorithm 1. It starts from $H' = \{v_0\}$ and then calls the $search$ function. The $search$ function exhaustively enumerates each vertex v form the neighbors of H' such that $\delta(H' \cup \{v\}) \geq \delta(H')$. If a solution is found, the procedure stops. Otherwise, it calls $search$ recursively. Theorem 2 ensures that such exhaustive enumeration can always find a valid solution to CST(k). The solution can be directly extended for CSM, but we omit the details here. Clearly, this baseline solution is of exponential complexity. This motivates us to develop a more efficient local search solution. Next, we will present our linear solution.

Algorithm 1 $Search()$

Input: $G(V, E)$, H', k
Output: H
1: **if** $\delta(G[H']) = k$ **then**
2: $H \leftarrow H'$
3: return
4: **end if**
5: **for all** vertex v in the neighbors of H' **do**
6: **if** $\delta(G[H' \cup \{v\}]) \geq \delta(G[H'])$ **then**
7: $Search(H' \cup \{v\})$
8: **if** $H \neq \emptyset$ **then**
9: return
10: **end if**
11: **end if**
12: **end for**

4.2 A Framework for Solving CST

In this section, we introduce a local search framework for CST. As outlined in Algorithm 2, on the high level, it contains three simple steps. First, we check if the graph meets the necessary condition of containing an answer to $\texttt{CST}(k)$. Second, we perform `candidateGeneration()`, that is, we explore from the vicinity of the query vertex, and generate candidate set C, which may contain a solution to the problem. In most cases, the second step will find a solution to CST, but if not, in the final step, we perform a global search (see Section 3) in the k-core of the subgraph induced by C to find the solution.

Algorithm 2 A General Framework of CST

Input: $G(V, E)$, v_0, k
Output: solutions to CST(k)
1: **if** $k >$ `upperBound`(G) **then**
2: **return**
3: **end if**
4: $C \leftarrow$`candidateGeneration`(G, v_0, k);
5: **if** no solution is found **then**
6: perform global search in the k-core of $G[C]$;
7: **end if**
8: **return**

Algorithm 2 is guaranteed to return a valid solution as long as `candidateGeneration()` does not remove any admissible vertex. This is summarized by Proposition 4.

PROPOSITION 4. *For graph G and query vertex v_0, if $H \subseteq V$ is a solution of $CST(k)$, then for any $H' \subseteq V$, a k-core of $G[H \cup H']$ that contains v_0 is a valid solution of $CST(k)$.*

In the following, we first introduce the upper bound in Section 4.2.1. We present a naive `candidateGeneration()` in Section 4.2.2 and we analyze its complexity and effectiveness in depth.

4.2.1 Upper Bound

Before we perform search, can we tell if a graph G has a solution for $\texttt{CST}(k)$ with respect to a query vertex v? Obviously, if the degree of v is less than k, we know immediately that there is no solution to $\texttt{CST}(k)$. In this section, we establish an upper bound of $m^*(G, v)$. If k is larger than the upper bound, then we immediately know there is no solution. The upper bound is given by Theorem 3.

THEOREM 3. *Given a connected simple graph $G(V, E)$, for any $v \in G$, we have*

$$m^*(G, v) \leq \lfloor \frac{1 + \sqrt{9 + 8(|E| - |V|)}}{2} \rfloor$$

PROOF. Since G is a connected graph, we have $|E| \geq |V| - 1$. For simplicity, we use H^* and m^* to denote the optimal community and its minimum degree. Then $G[H^*]$ has at least $\lceil m^*|H^*|/2 \rceil$ edges. For the rest of the vertices in $V - H^*$, there must be at least $|V| - |H^*|$ edges to ensure G is connected. Hence,

$$\lceil \frac{m^*|H^*|}{2} \rceil + |V| - |H^*| \leq |E| \quad (2)$$

It is also easy to see that $|H^*| \geq m^* + 1$. Thus, from Equation 2, we have $\lceil (\frac{m^*}{2} - 1)|H^*| \rceil \leq |E| - |V|$. By simple transformation, we have $(\frac{m^*}{2} - 1)(m^* + 1) \leq |E| - |V|$. Solving the above equation, we get $m^*(G, v) \leq \lfloor \frac{1+\sqrt{9+8(|E|-|V|)}}{2} \rfloor$. □

4.2.2 A Naive Candidate Generation Method

In this section, we present a straightforward implementation of `candidateGeneration()`. We give a more sophisticated solution in Section 4.3.1.

Algorithm 3 Naive `candidateGeneration()`

Input: $G(V, E)$, v_0, k
Output: C
1: $queue.enque(v_0)$; $C \leftarrow \emptyset$;
2: **while** $queue$ is not empty **do**
3: $v \leftarrow queue.deque()$;
4: $C \leftarrow C \cup \{v\}$;
5: **if** $\delta(G[C]) \geq k$ **then**
6: A valid solution is found and **Return** C;
7: **end if**
8: **for** $each(v, w) \in E$ **do**
9: **if** w is not visited and $deg_G(w) \geq k$ **then**
10: $queue.enque(w)$;
11: **end if**
12: **end for**
13: **end while**
14: **return** C

We show an example of this pseudo-code in Example 7. In this naive strategy, we generate candidate vertices by BFS starting from the query vertex v_0. As shown in Algorithm 3, we prune vertices of degree less than k as we traverse the graph (Proposition 3 of Section 2.2). Algorithm 3 runs in $\Theta(n' + m')$ time, where n' is the number of vertices and m' is the number of edges in $G[C]$.

4.2.3 Complexity Analysis

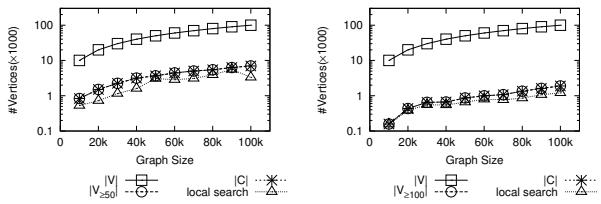
Next, we study the time complexity of Algorithm 2 when it implements `candidateGeneration()` as in Algorithm 3. The last step of the global search has the same complexity as that of candidate generation. Thus, the complexity of Algorithm 2 is $\Theta(n' + m')$. Hence, to reduce the complexity, we need to reduce $n' = |C|$ and m'. Section 4.3 proposes optimization techniques to reduce n' and m'. Before that, we first give the theoretic analysis of n' and m' to evaluate the pruning ability of the naive candidate generation.

Estimation of n'.

First, the value of n' has an obvious upper bound: $|V|$. In the worst case, we have $C = V(G)$. For example, when $k = N - 1$ and the graph is a complete graph with N vertices, for any query vertex, only the entire graph will qualify. Thus, in the worst case, the local search algorithm will not have any advantage over the global search algorithm.

Second, the value of n' has a tighter upper bound: $|V_{\geq k}|$. Let $V_{\geq k} = \{v | deg_G(v) \geq k\}$ be the set of vertices in G with a degree no less than k. Clearly, $G[C]$ belongs to one of the connected components of $G[V_{\geq k}]$. As an example, let G be the graph shown in Figure 1, and let g be the query vertex. Given $k = 4$, we have $C = \{g, h, i, j, k, l\}$, whose induced subgraph is a connected component of the subgraph induced by $V_{\geq 4} = \{d, e, g, h, i, j, k, l\}$.

We show in Figure 3 the simulation of above upper bounds of $|C|$ and the real size of $|C|$ found by the naive candidate generation. We generated scale free graphs of different sizes under the same parameters as that used in Section 6.2. We randomly select 10 vertices as query vertices and record the average metrics of interest. The size of the communities found by the improved local search proposed in Section 4.3.1 is also given (denoted by 'local search'). The simulation results show that $|C|$ and the real community size is quite close to $|V_{\geq k}|$ but far away from $|V|$. This implies that $|V_{\geq k}|$ is a good upper bound of C. In the estimation of m', we will use $|V_{\geq k}|$ as the approximation of $|C|$.



(a) $k = 50$ (b) $k = 100$

Figure 3: Simulation of upper bounds on $|C|$

Estimation of m'.

Clearly, the number of edges in $G[V_{\geq k}]$ is an upper bound of m'. Degree distribution is one of important feature to characterize a real network. We estimate the number of edges in $G[V_{\geq k}]$ based on the degree distribution of G.

Let p_k be the probability that a vertex chosen uniformly at random has degree k. Let $P = \{p_0, p_1, p_2, ..., p_\omega\}$ be the degree distribution of G, where ω is the largest degree of graph G, and $\sum_{0 \leq i \leq \omega} p_i = 1$. In general, it is reasonable to assume that $\omega \in o(|V|)$ for large graphs in real life. Given these assumptions, our major result is: *Given a graph G with degree distribution $P = \{p_0, p_1, p_2, ..., p_\omega\}$ and the maximal degree $\omega \in o(|V|)$, m' can be estimated by*

$$|V_{\geq k}| \sum_{t=1}^{\omega} t \times q_t = n \sum_{i=k}^{\omega} p_i \sum_{t=1}^{\omega} t \times q_t \tag{3}$$

where q_t is defined as Equation 4. This is built upon Theorem 4 and Lemma 5. Theorem 4 gives the degree distribution of $G[V_{\geq k}]$ (in an asymptotic case, that is, when the size of the graph is large enough). Lemma 5 gives the estimation of the largest degree in $G[V_{\geq k}]$. We omit their proofs due to space limitations.

THEOREM 4. *Given a graph G with degree distribution $P = \{p_0, p_1, p_2, ..., p_\omega\}$ and the maximal degree $\omega \in o(|V|)$. Let q_t be the probability that a vertex chosen uniformly at random in $G[V_{\geq k}]$ has degree t. Then we have:*

$$q_t = \sum_{i=t}^{\omega} p_i \binom{i}{t} p^t (1-p)^{(i-t)} \tag{4}$$

where $p = \frac{\zeta(k)}{\zeta(0)}$ and $\zeta(x) = \sum_{i=x}^{\omega} i \times p_i$.

LEMMA 5. *For a graph G with the largest degree ω and degree distribution $P = \{p_0, p_1, p_2, ..., p_\omega\}$. It holds* asymptotically almost surely *that the largest degree of $G[V_{\geq k}]$ is ω as $n \to \infty$, where n is the number of vertices in G.*

4.3 Optimization

In this section, we introduce two optimization techniques that improve the performance of the algorithm by reducing n' and m'.

4.3.1 Intelligent Candidate Selection

Algorithm 3 is naive in the sense that it blindly chooses vertices from the queue. The following example shows that it may take the naive approach more steps to find a solution.

Step1: C={e}	Q={a,c,d,f}
Step2: C={e,a}	Q={b,c,d,f}
Step3: C={e,a,f}	Q={b,c,d,g,n}
...	
Step12: C=V-{n,m}	Q={}

(a) Naive selection

Step1: C={e}	Q={a,c,d,f}
Step2: C={e,a}	Q={b,c,d,f}
Step3: C={e,a,d}	Q={b,c,f}
Step4: C={e,a,d,b}	Q={c,f}
Step5: C={e,a,d,b,c}	Q={f}

(b) Intelligent selection

Figure 4: Naive vs. intelligent candidate generation

EXAMPLE 7 (CANDIDATE GENERATION). *Consider the graph shown in Figure 1. For query vertex e and $CST(3)$, using the naive candidate generation method, f may be added into C (as shown in the Step 3 of Figure 4(a)), which leads to no valid solution. But the procedure still proceeds until all vertices are exhaustively enumerated (overall 12 steps are needed). On the other hand, if we always choose the vertex with the largest number of connections to C, a valid solution can be found within 5 steps. The procedures for these two selections are illustrated in Figure 4, where Q is the queue from which the vertex is selected.*

To reduce $n' = |C|$, we propose two intelligent candidate selection strategies. The basic idea to refine the candidate generation is using *priority queue* so that we can select the most *promising* vertex that leads to a solution fast. Theoretically, Lemma 3 can be used to compute the "promisingness" of a vertex. However, its computation is too costly. We are interested in lightweight heuristics of constant cost. Next, we will propose two intelligent lightweight heuristics.

Largest increment of goodness (**lg**). Selecting the vertex that leads to the largest increment of the goodness measure is a straight-forward heuristic since the final goal of CST is to find a subset C satisfying $\delta(G[C]) \geq k$. In this strategy the priority of a vertex v, $f(v)$, is defined by

$$f(v) = \delta(G[C \cup \{v\}]) - \delta(G[C]) \tag{5}$$

It is a greedy approach as it only considers the improvement of $\delta(C)$ for the next step. Note that at any time when a vertex is added to C, the increment of the goodness of the current C is at most 1. In other words, for any v, we have $f(v) = 1$ or $f(v) = 0$. Hence, this strategy is equivalent to the random selection from vertices which are adjacent to one of the vertices with the minimal degree in C.

Largest number of incidence (**li**). This is a more intelligent selection strategy. The priority of a vertex v is defined as

$$f(v) = deg_{G[C \cup \{v\}]}(v) \tag{6}$$

In this strategy, we select the vertex with the largest number of connections to the selected vertices. This yields the fastest increase of the mean degree of $G[C]$. In general, the lowest degree of a graph will increase with the growth of its density, and consequently a valid solution C with $\delta(G[C])$ is expected to be found within a limited number of steps if there exists such a solution.

As we have shown in Example 7, these selection strategies generally are quite efficient. However, Example 8 shows that local information generally is not sufficient to construct a valid solution within the WHILE loop. In this case, we need to perform a global search in the k-core of $G[C]$ (line 6) of Algorithm 2. This step ensures a valid solution, as stated in Proposition 4.

EXAMPLE 8 (HARDNESS OF SELECTION). *Continue Example 7, by* **li***, it is quite possible that f is selected after e. As a result, even after all vertices are visited, no solution can be found.*

Complexity Analysis

Let n' and m' denote the number of vertices and the number of edges in $G[C]$, respectively. In general **lg** and **li** can be implemented in $O(n' + m' \log n')$ time, because each time a new vertex is added into the queue, at most d queue update operations need to be executed (where d is the degree of this vertex). As a result, at most m' update operations will be executed. Each queue operation (insert, delete, or priority update) generally needs $O(\log n')$ time.

By careful design, **li** can be implemented in $O(n' + m')$, achieving $O(1)$ cost of expansion. We maintain a collection of lists each of which, contains vertices of V with the same $f(v)$. Each time a vertex is added to C, the $f(\cdot)$ value of its neighbors (except those already in C) will *increase* 1. For each of these influenced neighbors, we move v from the original $f(v)$'s list to $f(v) + 1$'s list. In this manner, we can always find one vertex with maximal $f(\cdot)$

in $O(1)$ time. We illustrate the above procedure in Figure 5 and Example 9.

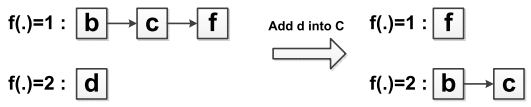


Figure 5: An example of the data structure used in li heuristic.

EXAMPLE 9. *Consider the example graph in Figure 1. Suppose we have added e and a into C (candidate vertices). Then we have $f(b) = f(c) = f(f) = 1, f(d) = 2$. According to the $f()$ function value (i.e. the number of incidence to C), we can organize C's neighbors into two different lists, which are shown in the left part of Figure 5. We also record a pointer to the list with the maximal incidence. The pointer helps find that node d has the most incidences to C (Line 3, Algorithm 3). Then, we move d from its list into C (Line 4, Algorithm 3). And update the two lists by moving b and c from the list $f(.) = 1$ to the list $f(.) = 2$. In this way, we keep the freshness of the lists.*

4.3.2 Intelligent Expansion

We have created a strategy to prune the neighbors we need to visit, i.e. reducing m'. The basic idea is to sort the adjacent list of vertices. We use the adjacent list to represent a graph and sort the adjacent list of each vertex into the descending order of degree. Then, during candidate generation, when we expand a vertex's neighbors (Line 9-11 in Algorithm 3), we stop the expansion immediately when the neighbor has a degree of less than k. Consequently, we can avoid the cost of scanning all neighbors.

For a vertex with d neighbors, we need $O(d \log d)$ time to order its adjacent list. To avoid this cost, we perform the ordering as a pre-computation step before online query processing. In real life applications such as an online social network, numerous community queries may be issued by users. In these cases, precomputation to order the adjacent list is desired. When the graph is dynamically evolving, maintaining the order of adjacent lists is of marginal cost. We can use a binary search tree to represent the ordered adjacency list, costing us only $O(\log(d))$ each time the list changes.

Clearly, this optimization is only effective for local search and cannot be used to speedup global search. Our experiments show that this optimization boosts local search significantly.

5. LOCAL SEARCH FOR CSM

The goal of CSM is to find the best community for a given vertex. The challenge is that the goodness measure $\delta(\cdot)$ is not monotonic. In Section 4.1, we have shown an exhaustive enumeration approach. But it is of exponential complexity. In Section 2.2, we introduced a method for solving CSM by repeatedly calling CST. In this section, we devise a more efficient, bottom-up solution. The algorithm takes 3 steps. First, it expands the search space from the query vertex v_0. Second, it generates a candidate vertex set C in the search space. Third, it invokes the maximum core method to find the final solution in the candidate set.

5.1 Expanding the Search Space

In this step (line 1 to 15 of Algorithm 4), our goal is to expand the search space and find a subset H whose $\delta(G[H])$ is as large as possible while pruning as many invalid vertices as possible under the linear cost. We start with v_0, and at each step we select the vertex that is the local optimal and add it into the current result set. Here, we use the **li** (largest number of incidence) strategy to select the local optimal vertex (line 6 to 7). Then, at the end of each iteration, we know that any vertex whose degree is smaller than $\delta(G[H]) + 1$ cannot appear in any better solution. Thus, in line 14, we expand the set of vertices we need to visit by using H for filtering.

Algorithm 4 A local-search framework to solve CSM

Input: a graph $G(V, E)$, v_0 , $-\infty < \gamma < \infty$
Output: H

```
    {Step 1: Iterative searching and filtering.}
 1: H ← ∅ /* the best solution found so far */
 2: A ← {v0} /* vertices we have visited */
 3: B ← {v|(v, v0) ∈ E} /* vertices we need to visit */;
 4: s ← 0
 5: while B ≠ ∅ and s ≤ e^{-γ}(⌊(|E|-|V|)/(δ(G[H])+1)/2-1⌋ - |H|) do
 6:    Let v be the vertex with most links to A from B;
 7:    s ← s + 1; A ← A ∪ {v}; B ← B − {v};
 8:    if δ(A) > δ(H) then
 9:       H ← A;s ← 0;
10:       if δ(H) = min{deg_G(v0), ⌊(1+√(9+8(|E|-|V|)))/2⌋} then
11:          Return H;
12:       end if
13:    end if
14:    Add v's neighbors with degree larger than δ(G[H]) into B;
15: end while
    {Step 2: Generating candidates}
16: C ← Generate candidate vertex set based on H or A;
    {Step 3: Finding the solution}
17: H ← maxcore(G[C], v0);
18: return H;
```

One critical question is when we can stop expanding the search. Clearly, H is the optimal solution if

$$\delta(H) = \min\{deg_G(v_0), \lfloor \frac{1 + \sqrt{9 + 8(|E| - |V|)}}{2} \rfloor\} \quad (7)$$

However, Equation 7 is a *sufficient but not necessary* condition. For instance, if an invalid vertex is introduced into H in the early stage of expansion, H may never reach the upper bound, even if it already contains the optimal solution. To solve this problem, we introduce another upper bound. We consider the extra number of vertices that need to be added into the current H in order to improve the minimum degree of $G[H]$, given that a better solution that contains the current H exists. This upper bound is given by Corollary 1, which is derived from Theorem 5.

THEOREM 5. *Let $G(V, E)$ be a connected graph, and $v \in V$ be the query vertex. If H is a solution to $\texttt{CST}(k)$, we have*

$$|H| \leq \lfloor \frac{|E| - |V|}{k/2 - 1} \rfloor$$

PROOF. Since G is connected, we have $|E| \geq |V| - 1$. Also, $G[H]$ has at least $k|H|/2$ edges. There exist at least $|V| - |H|$ edges incident with vertices in $V - H$ to maintain the connectivity of graph G. Hence, we have $k \cdot |H|/2 + |V| - |H| \leq |E|$, which leads to $|H| \leq \frac{|E|-|V|}{k/2-1}$. □

COROLLARY 1. *Let $G(V, E)$ be a connected graph, and let H be the current optimal solution found so far in Algorithm 4, if there exists $H' \supset H$ such that $\delta(G[H']) = \delta(G[H]) + 1$, we need to add at most*

$$\lfloor \frac{|E| - |V|}{(\delta(G[H]) + 1)/2 - 1} \rfloor - |H|$$

vertices to find H'.

From Corollary 1, we can see that the larger $\delta(G[H])$, the tighter the upper bound. So this upper bound has better pruning power when the solution has larger m^*.

Given the above upper bound, we use two parameters s and γ (the latter is exposed to users) to control the searching space. The value of s denotes the number of vertices that have been added to H (see line 7 and line 9). In line 5, we use

$$s \leq e^{-\gamma}(\lfloor \frac{|E| - |V|}{(\delta(G[H]) + 1)/2 - 1} \rfloor - |H|) \quad (8)$$

with $-\infty < \gamma < \infty$ to control the search space. We terminate the search when the upper bound is reached. Note that when $\gamma = 0$, Eq 8 degrades into the exact bound given in Corollary 1. When $\gamma > 0$, the number of extra vertices that will be added into H will be less than the exact bound. When $\gamma \to -\infty$, no constraint on s is specified. We will come back to these parameters after the introduction of following steps of the algorithm.

5.2 Generating Candidates

In the second step, we generate a candidate set from H obtained in the first step. Specifically, we propose two solutions to generate C (line 16), and we analyze the tradeoff between their quality and performance. Let $C_{naive}(k)$ be the result of Algorithm 3. That is, $C_{naive}(k)$ is the set of vertices obtained by iteratively removing vertex of degree less than k from the neighborhood of vertex v_0.

Solution 1: $C \leftarrow A$. In this case, we have the following result:

THEOREM 6. *Given graph G and a query vertex v_0, when $\gamma \to -\infty$, Algorithm 4 finds an optimal solution for* CSM.

PROOF. Consider when the WHILE loop exists in Algorithm 4. Let $k = \delta(G[H])$. Clearly, we have $m^*(G, v_0) \geq k$. We have $C_{naive}(k) \subseteq A$. Then, we have $C_{naive}(m^*(G, v_0)) \subseteq C_{naive}(k)$. Thus $C_{naive}(m^*(G, v_0)) \subseteq A$. Thus $maxcore(G[A], v_0)$ is an optimal solution. □

When $C \leftarrow A$, Algorithm 4 also allows us to trade quality for performance by tuning parameter γ. Specifically, when γ is closer to $-\infty$, the solution is of higher quality; when γ is closer to ∞, better performance can be achieved. We will discuss more about this tradeoff in the experiments section.

Solution 2: $C \leftarrow C_{naive}(k)$. Here, $k = \delta(G[H])$.

THEOREM 7. *Given graph G and query vertex v_0, Algorithm 4 always finds an optimal solution: $maxcore(G[C_{naive}(k)], v_0)$, where $k = \delta(G[H])$.*

PROOF. For a graph G and a query vertex v_0, $C_{naive}(k)$ becomes larger when k becomes smaller. Hence, for any k with $\leq m^*(G, v_0)$, $C_{naive}(k)$ contains all valid vertices. Since Algorithm 4 will only return a subset $H \subseteq V$ with $k = \delta(G[H])$ no larger than $m^*(G, v_0)$, $maxcore(C_{naive}(k), v_0)$ is the optimal solution of $\text{CSM}(G, v_0)$. □

Note that the choice of γ has no influence on the quality of the solution, but it may influence the running time. In general, when $\gamma \to -\infty$, it is quite possible to find a complete solution or at least a good partial solution before the second step since we have given enough number of tries to find a good H. However, this comes at high cost of run time. If a solution is already found, or a partial solution is found in the first step, the maxcore procedure will be unnecessary or use little time. Hence, it is possible for the existence of an optimal γ leading to the minimal running time. We will show this in the experiment section.

Finally, we highlight that Algorithm 4, under a different implementation strategy, runs in $O(|V| + |E|)$ time in the worst case. However, in general, similar to CST, in CSM a smaller number of vertices are expected to be visited for an optimal solution in our local-search solutions than their global search competitors.

6. EXPERIMENTS

In this section, we use experiments on real life graphs to show the effectiveness and efficiency of our local search method. We also conduct experiments on synthetic graphs to show its scalability. We implemented all of the algorithms in C++ and ran the experiments on a PC with AMD Athlon^{TM}X2 Dual-core QL-62 at 2GHz, 2G memory.

6.1 Experiments on Real Life Graphs

6.1.1 Datasets

We used four real life large graphs in our experiments: DBLP, Berkeley, Youtube, and LiveJournal. DBLP is an author collaboration network, where each vertex represents an author and each edge represents a coauthor relationship. Berkeley is a web graph with nodes representing pages from berkeley.edu and stanford.edu domains and edges representing hyperlinks between them. We ignored the direction of the links. Youtube [11] is a user-to-user link network. LiveJournal is an online social network. For each dataset, we only consider the largest connected component. The statistics of these graphs are reported in Table 2, where the minimal vertex degree of the maximal core of this graph is given as $\delta^*(G)$.

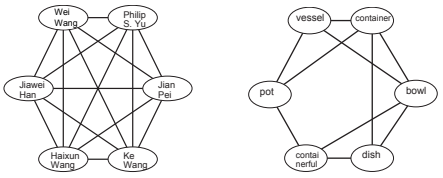
Network	#Vertex	# Edge	$\delta^*(G)$	Opt.(ms)	k=20	40	60
DBLP	481K	1.72M	114	703	0	0	0
Berkeley	654K	6.58M	202	2328	9	0	0
Youtube	1.1M	3M	52	1359	0	0	0
LiveJournal	4.0M	34.7M	360	2381	0	0	0

Table 2: Basic information of real dataset

6.1.2 Case Study

We presented two case studies to justify the minimal degree based community search. The first is on DBLP. We used "Jiawei Han", who is a renowned scientist in Data Mining, as the query vertex. After setting $k = 5$, we got the community shown in Figure 6(a) using $ls - li$. We found that these six authors are all leading scientists in data mining community and their cooperation is very frequent. For example, Jiawei Han and Jian Pei have coauthored over 37 papers, and Haixun Wang has coauthored with Philip S. Yu and Jian Pei over 46 and 15 papers, respectively.

The second case study is on WordNet. WordNet is a semantic network, in which each vertex represents a specific sense, and each edge represents a certain semantic relationship between senses. In this case study, we used the word *pot* as v_0. Using $ls - li$ with $k = 3$, we got the community shown in Figure 6(b). We found that the senses in the community are highly related in semantic: all of them are about the vessel. *pot*, *bowl*, *dish* are some vessel entity. *vessel* and *container* are two hypernyms of those entities. And *containerful* is an adjective word related to containers.



(a) Jiawei's community (b) Community for "Pot"

Figure 6: The effectiveness of community search.

6.1.3 Results for CST

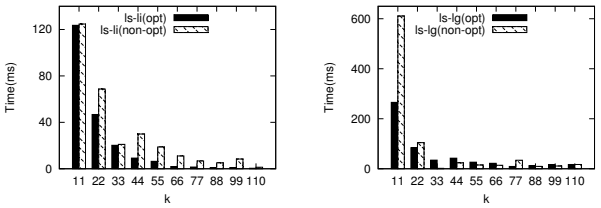
We evaluate the efficiency of our local search solutions for CST by comparing it with the global search solution introduced in Section 2. We denote the global search as **global**, and we denote the three versions of local search, i.e., the local search with naive candidate generation, **li**, and **lg** as **ls-naive**, **ls-li** and **ls-lg** respectively.

Solutions to CST are sensitive to the input parameter k. To evaluate the performance of different solutions, we tested each method with $k = s, 2s, 3s, ..., 8s$, where $s = \delta^*(G)/10$. For each k, we randomly selected 100 vertices as the query vertex from the k-core of the graph so that there was always a meaningful solution containing the query vertex. Then, we averaged the query time of these 100 query vertices.

Baseline solution. We first showed that the baseline solution in general is costly on big graphs. For each real network, we selected $k = 20, 40, 60$. For each k, we randomly selected 100 vertices

with a degree of no less than k as the query vertex. We recorded in the last three columns of Table 2 the number of vertices for which the query result can be returned within 1 minute. The results show that in most cases the baseline solution cannot produce result in the given time. Hence, in the following experiments, it will be omitted.

Efficiency of off-line ordering. We first justified the optimization of local search by ordering the adjacency list. We just give the results on DBLP data. Similar results on other real life graphs were obtained and omitted to save space. In Figure 7, we compare local search with optimization (opt) to local search without optimization (non-opt). We can see that, for both **ls-li** and **ls-lg** under most parameter values of k, the optimization technique brings an obvious speedup. The optimization technique has a linear cost as it only requires sorting the adjacent list of each vertex. For example, as reported in Table 2, for DBLP data, it only incurs 703ms. Hence, in all the following experiments, all local search solutions are optimized in this way.



(a) ls-li (b) ls-lg

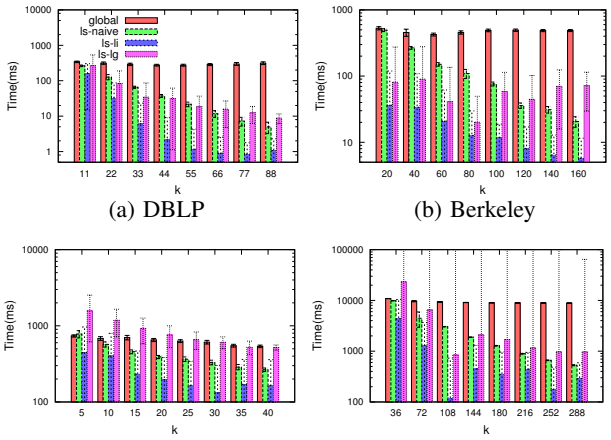
Figure 7: The effectiveness of the optimization technique.

Performance for CST. We show the performance (mean running time as well as its std) of local search solutions for CST in Figure 8. We can see that, in general, local search performs better than global search in most cases. When k increases, the advantage of local search over global search becomes more obvious. In the best case, for example when k is large on DBLP and Berkeley, ls-li or ls-lg is two orders of magnitudes faster than global search. Only when k is small, global search is comparable to naive local search and the two optimized local search solutions. The reason is that when k is small, most vertices in the graph tend to be involved in the answer, which favors global search. We can also see that among all strategies of local search, in most cases, **ls-li** performs the best. The running time for **ls-li** almost monotonically decreases with the growth of k. This implies **li** is quite effective when k is large.

The stds show that the global approach is the most stable algorithm. This is because the search space of the global algorithm is always the entire graph. In contrast, the local search solutions need to search the entire graph only when the search within the local neighborhoods fails. That is to say, in most cases, local search solution returns results quite fast. Only in some rare cases, its running time is close to global solution (but will not be worse than global search). That is why our solution has larger stds. From the experimental results, we also can see that even considering the worst case, that is *mean running time+ std*, $ls - li$ is better than $global$ in most cases.

In some rare cases, for example in Figure 7(c), the global algorithm is better than $ls - lg$ or $ls - naive$. The reason is that $ls - lg$ and $ls - naive$ are not very intelligent. Their search spaces although are less than the global one, in many cases are close to the global one. Furthermore, in general local search solutions need to select the best vertex from candidates, which adds extra computation cost. Consequently, $ls - lg$ or $ls - naive$ may consume more running time. Similar results can be found from Figure 8(b), 8(c) and they can be explained similarly

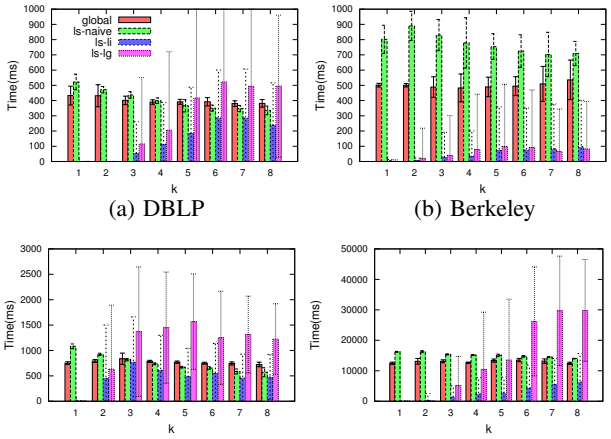
Influence of small k on local search. To take a close look at the influence of small k (from 1 to 10), we conducted more experiments, the results of which are shown in Figure 9. It is quite interesting to note that when k is extremely small, local search is



(a) DBLP (b) Berkeley

(c) Youtube (d) LiveJournal

Figure 8: The efficiency of different solutions to CST



(a) DBLP (b) Berkeley

(c) Youtube (d) LiveJournal

Figure 9: Performance of CST for small k

significantly better than global search (it outperforms global search by two orders of magnitude). This is because when k is extremely small, local search tends to find smaller communities. For example, when $k = 1$, any edge incident with the query vertex will be immediately returned as an answer; when $k = 2$, any cycle containing the query vertex is an answer. In summary, local search is consistently better than global search across a wide range of k.

Performance over arbitrary vertices. In the above experiments, the query vertices come from k-core, which means that a valid community certainly exists. In this experiment, we tested the performance over arbitrary vertices for which a valid community does not necessarily exist. We randomly select 100 query vertices whose degree is no less than k from DBLP (otherwise we can surely find no community for these vertices). We compare the best of local search solutions $ls - li$ to the global search. The result is shown in Figure 10. Similar to previous experiment results, local search is better than global search in almost all the cases. When k grows, the search space of local search becomes smaller. As a result, the mean running time of local search decreases. Instead, the global search is not aware of k and consistently consumes almost the same running time.

Rationale of local search. To understand the rationale of local search, we report the answer size and the number of visited vertices in the search. We show the results for DBLP in Figure 13 (results on other real networks are similar and are omitted to save space). We can see that the local search method tends to produce a small community. In some cases, the community found by **ls-li** or

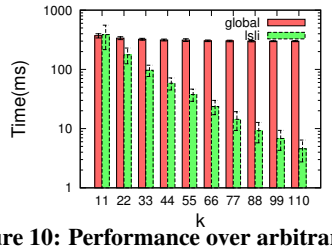


Figure 10: Performance over arbitrary vertices

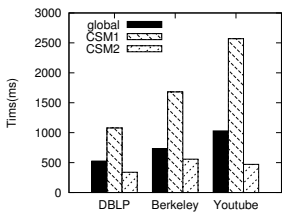


Figure 11: The performance of local search to CSM

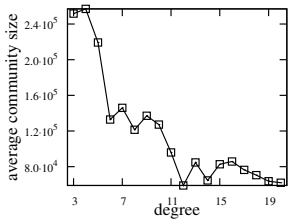


Figure 12: Community size vs vertex degree on DBLP

ls-lg is an order of magnitude smaller than that produced by global search and **ls-naive**. The comparison of the number of visited vertices shows that local search visits a much smaller number of vertices than global search. In many cases, local search outperforms global search by two orders of magnitude. Both smaller answer sizes and a fewer number of visited vertices explain the advantage of local search over global search.

6.1.4 Results for CSM

Recall that there are two local search solutions to CSM. We denote the one that generates C by $C \leftarrow A$ as $CSM1$, and the other as $CSM2$. We also compare them with **global**. In local search, we present the results of **li** only, since previous results have already shown its efficiency and effectiveness.

Performance for CSM. For $CSM1$, we can trade the quality of the solution for performance by tuning parameter γ. For the sake of fair comparison, we set $\gamma \rightarrow -\infty$ such that $CSM1$ is not constrained by size s. The result is shown in Figure 11. From the comparisons, we can see that $CSM2$ performs the best. $CSM1$ consumes the most time since we remove the size constraint of s, which means that the search procedure will exhaustively search a huge space. In the next experiment, we will show that by tuning γ, $CSM1$ can run much faster than global search without sacrificing the quality.

Effect of γ on $CSM1$. For $CSM1$, γ controls the tradeoff between quality and performance. We ran $CSM1$ with γ varying from 1 to 15 to observe the change in quality and run time. The quality of $CSM1$ is measured by

$$r_a = \Sigma_{v_0}\delta(H')/\Sigma_{v_0}\delta(H)$$

, where H' is the answer found by $CSM1$, H is one of the optimal answers found by a global search, and the Σ runs over all randomly selected query vertices. We also summarize the time efficiency measured by

$$r_t = \Sigma_{v_0}t_1(v_0)/\Sigma_{v_0}t_2(v_0)$$

, where $t_1(v_0)$ is the run time of $CSM1$ for query vertex v_0, $t_2(v_0)$ is the run time of the global search, and the summarization runs over all sampled query vertices.

The tradeoff between quality and performance is shown in Figure 14. As expected, both performance and quality decrease with the growth of γ. However, performance drops drastically faster than quality. As a result, even running for a short duration, the quality produced by $CSM1$ still remains almost at one hounded of percent, especially for DBLP and Berkeley. For each of the three networks, we can observe a critical point (identified by the dotted line), before which a minor drop of solution quality will bring us a significant improvement of performance. The above results clearly show that the trading of quality for performance in $CSM1$ is quite effective. In real applications, users can specify γ based on their requirement for the tradeoff between performance and accuracy.

Effect of γ on $CSM2$. From Figure 15, we observe that for each graph there is a certain γ that leads to the minimal overall run time of CSM2. The influence of γ on the performance of $CSM2$ generally depends on the network structure. It seems that Berkeley is more sensitive to γ than the other two networks. However, our results show that generally when $4 \leq \gamma \leq 12$, minimal run time of $CSM2$ can be achieved.

Selection of γ. Next, we give guidelines for selecting an appropriate γ for a query vertex when solving CSM. In general, the selection of γ depends on the size of the resulting community, which is determined by the local structure near the query vertex. The larger the community size, the larger the search space. Hence, γ should be tuned accordingly. However, it is hard to know the exact community size before a local search is executed. We found that, through the empirical analysis of DBLP data, *the degree of the query vertex is a good hint of the vertex's community size*. We used global search to find a maximal community for a query vertex in DBLP. For each degree, we randomly selected 10 vertices and recorded their average community size. The results are shown in Figure 12. We observe that the community size generally decreases when the degree of the query vertex increases. Hence, we should choose a smaller γ when the query vertex has a large degree (recall that smaller γ leads to larger search space).

6.2 Experiments on Synthetic Networks

To test the scalability and the sensitivity-to-community-structure of our solutions, we generated a collection of synthetic networks. Community search is meaningless on a network without community structures, hence we used a generator [12] that can generate synthetic graphs with a varying degree of "clearness" of community structure. It generates a graph using three parameters α, β, and μ. The degree and the community size follow the power-law distribution with exponent $-\alpha$ and $-\beta$, respectively. Parameter μ indicates the proportion of a vertex's neighbors that reside in other communities. Clearly, by tuning μ, we can vary the clearness of the community structure of the generated synthetic network. Since **ls-li** is the most efficient local search method, in the following experiment, the results of the local search are produced by **ls-li**.

Scalability. Using the above network model, we generated an ensemble of synthetic networks with size varying from $200K$ to $1M$ with $\alpha = 2, \beta = 3, \mu = 0.1$.

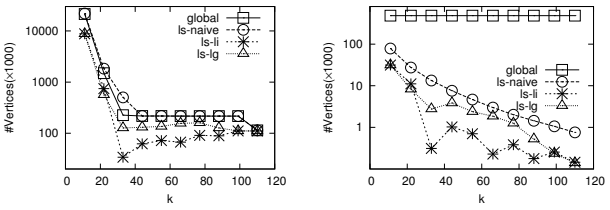
The scalability results of the local search are shown in Figure 16. We found that **ls-li**, $CSM1$, and $CSM2$ are consistently more efficient than a global search even on graphs with millions of vertices. Note that the result for $CSM1$ was obtained with 100% accuracy (we omitted the accuracy result). The efficiency of $CSM1$ is quite impressive since it consistently outperforms a global search by about three orders of magnitude without sacrificing the accuracy. We also found that the run time of local search grows more slowly than global search when the graph size increases. In some cases, for example in CST, the advantage of a local search over global search is even more obvious when the graph is larger. These observations verify that a global search needs to visit all vertices. In contrast, a local search only needs to visit the limited neighborhoods around the query vertex. When the neighborhood grows more slowly than the whole graph size, a local search is certainly more efficient than a global search.

Sensitivity to community structure. Local search is sensitive to the clearness of community structures in the network. The more obvious the community structure, the more efficient local

search. To verify this conjecture, we generate synthetic networks with μ varying from 0.1 to 0.5. Other parameters were the same as the previous experiment. The larger the value of μ, the clearer the community structure.

We present the result in Figure 17. It is clear that **ls-li** and $CSM1$ are consistently and significantly better than global search for different μ. $CSM2$ exhibits a similar performance as global search, but is still better than global search. The results in Figure 17(c) show that $CSM1$ can help us find a good tradeoff between quality and performance, independent of the community structure of the network.

We can also see that either the local search (both solutions of CST and CSM) or global search consumes more time when the network shows vaguer community structure. When μ is large, the boundary of the communities become vague, thus, for most of k, the result of $CST(k)$ that contains the query vertex will become larger, and the maximal core of the graph tends to become larger. As a result, for both local search and global search, the run time increases with the growth in the answer size.



(a) The answer size (b) The number of visited vertices

Figure 13: Answer size and visited vertex size in local search

7. RELATED WORK

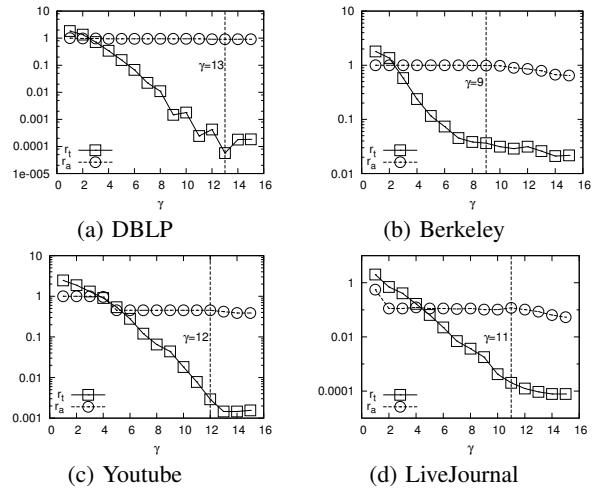
The work presented in this paper is closely related to *community detection*, *local search on graphs*, and *k-core decomposition*.

Community detection. Community detection was first modeled as a *modularity* optimization problem. *Modularity* [1] is a well-known measure of the *goodness* of non-overlapping divisions in a network. Unfortunately, finding the division that maximizes the modularity is NP-complete [13]. A typical solution to optimizing modularity is the Newman-Girvan approach [1], which derives good partitions on many real networks but it takes $O(|E|^2|V|)$, or $O(|V|^3)$ time on a sparse graph. It was also shown that modularity optimization may fail in some special networks [14].

Many successors devoted their efforts to improving the quality and performance of community detection. In recent years, label propagation has attracted wide research interest since it in general can produce high-quality communities with almost linear time [15, 16]. In real applications, only top clusters usually interest users. A solution [17] was proposed to mine the top clusters of a network. Sparsification that can preserve the community structure is also used to reduce the size of the graph so that community detection on the sparsified graph is more efficient [18].

Another direction of efforts focuses on extending current solution to support more semantics of communities. The real social networks usually have overlapping communities [19], *i.e.*, vertices are shared among different communities. Some efficient approaches for the identification of overlapping communities in large real network have been proposed, including clique percolation method [20], q-state Potts model based method [21]. Considering the influence of attributes of edges on the semantic of community, SA-Cluster, which utilizes both structure and attribute information, is proposed for graph clustering [22].

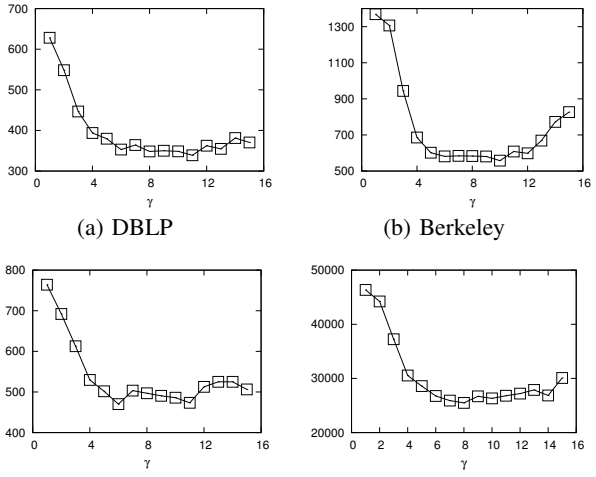
All these works do not consider *community search*, which was first proposed in [5]. The community search problem addressed in this paper is a special case of that proposed in [5], where they intend to find a community containing a set of query nodes while



(a) DBLP (b) Berkeley

(c) Youtube (d) LiveJournal

Figure 14: γ's affect on $CSM1$



(a) DBLP (b) Berkeley

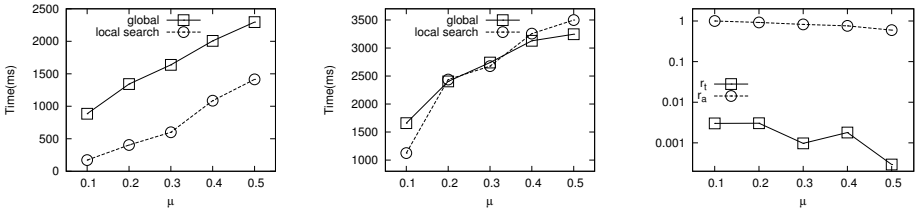
(c) Youtube (d) LiveJournal

Figure 15: γ's affect on $CSM2$

in our problem we are interested in querying the community for a single vertex. However, they solve the community search problem by global search.

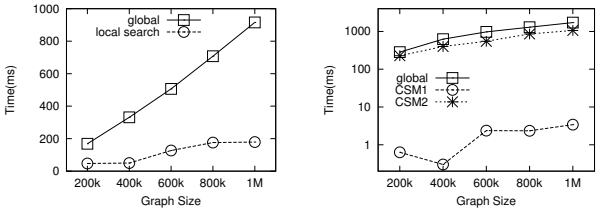
Local search for communities. In recent years, local search for communities in real graphs is attracting ever-increasing research interests [29, 30, 31, 32]. Aaron [29] first proposes the problem to find a community with size constraint k for a certain vertex. But he uses "local modularity" as the community goodness measure, which characterizes the relative density within the community to outside of the community. According to the new measure, he proposes a heuristic algorithm with quadratic time complexity regarding to k. Bagrow [30] further improves the performance by selecting the vertex with largest "outwardness", where outwardness of vertex v is the number of v' neighbors outside the community minus the number inside. Local search of community is widely used in existing Sybil defense schemes [31]. Their basic idea is that local community around a trusted node is also trustworthy [31]. And Wanyun [32] studies the overlapping structure of local search.

These local search methods [29, 30] find communities with size constraint to limit the search space. As a result, they cannot ensure to find the best community under corresponding community goodness measure. Another weakness is that the size constraint as



(a) CST (b) CSM2 (c) CSM1

Figure 17: Influence of community structure on efficiency of local search



(a) CST (b) CSM

Figure 16: Scalability of local search

an input parameter is hard to select. In general, each vertex has its own the most appropriate community size. Either predefining a global parameter or trying different parameters blindly can hardly find meaningful results.

k-core decomposition. The technique used in this paper is closely related to the estimation of giant component size of random graphs [23, 24] and k-core decomposition [25, 26]. The threshold of sudden emergence of a giant k-core in a random graph is theoretically studied in [25]. Newman *et al.* [24] proposed a novel method to compute the mean component size of random graphs with arbitrary degree distribution using generating function. k-core decomposition is empirically investigated on many real large networks and shown to be a new perspective to characterize the structure and function of real networks [27]. k-core decomposition is also used to visualize large complex networks [26]. An external memory based k-core decomposition solution is developed to handle massive networks [28]. These techniques are rarely used for community search.

8. CONCLUSION

We investigate the problem of finding the best community containing a given query vertex in its neighborhood. We propose a local search method for this purpose. Local search is more efficient than global search since global search needs to visit all vertices in the network for community detection. We address the local search challenge that arises from the non-monotonicity of community goodness measure. In this paper, we propose the CST and the CSM algorithms to solve a variety of community search problems. We conduct extensive experiments on synthetic and a variety of real life, million-node networks. Our future work will consider constraints in community search, so that we can support applications in many emerging social settings, such as micro-blogging and online social networks.

9. REFERENCES

[1] M. E. J. Newman et al., "Finding and evaluating community structure in networks," *Phys. Rev. E*, vol. 69, no. 2, p. 026113.

[2] A. Broder et al., "Graph structure in the web," *Comput. Netw.*, vol. 33, no. 1-6, pp. 309–320, June 2000.

[3] G. Palla et al., "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818.

[4] R. Guimera et al., "Functional cartography of complex metabolic networks," *Nature*, vol. 433, no. 7028, pp. 895–900.

[5] M. Sozio et al., "The community-search problem and how to plan a successful cocktail party," in *KDD'10*.

[6] H. Zhuge et al., "Query routing in a peer-to-peer semantic link network," *Computational Intelligience*, vol. 21, no. 2, pp. 197–216.

[7] B. Bollobás, "The evolution of sparse graphs," Graph Theory and Combinatorics, Proc. Cambridge Combinatorial Conf. in honor of Paul Erdos, Academic Press, pp. 35-57.

[8] M. Gaertler et al., "Dynamic analysis of the autonomous system graph," in *IPS 2004*, pp. 13–24

[9] S. B. Seidman., "Network structure and minimum degree," *Social Networks* , Vol. 5, pp. 269–287.

[10] S. N. Dorogovtsev et al., "k-core organization of complex networks," *Phys. Rev. Lett.* Vol.96, pp. 040601.

[11] A. Mislove et al., "Measurement and analysis of online social networks," in *IMC'07*.

[12] A. Lancichinetti et al., "Benchmark graphs for testing community detection algorithms," *Phys. Rev. E*, vol. 78, no. 4, p. 046110.

[13] U. Brandes et al., "On modularity clustering," *TKDE*, vol. 20, no. 2, pp. 172–188.

[14] S. Fortunato et al., "Resolution limit in community detection," *PNAS*, vol. 104, no. 1, p. 36.

[15] U. N. Raghavan et al., "Near linear time algorithm to detect community structures in large-scale networks," *Phys.Rev.E*, vol. 76, p. 036106.

[16] I. X. Y. Leung et al., "Towards real-time community detection in large networks," *Phys.Rev.E*, vol. 79, p. 066107.

[17] K. Macropol et al., "Scalable discovery of best clusters on large graphs," in *VLDB'10*.

[18] V. Satuluri et al., "Local graph sparsification for scalable clustering," in *SIGMOD '11*.

[19] G. Palla et al., "Uncovering the overlapping community structure of complex networks in nature and society," *Nature*, vol. 435, no. 7043, pp. 814–818.

[20] I. Derenyi et al., "Clique percolation in random networks," *Phys. Rev. Letters*, vol. 94, no. 16, p. 160202.

[21] J. Reichardt et al., "Detecting fuzzy community structures in complex x networks with a potts model," *Phys. Rev. Letters*, vol. 93, no. 21, p. 218701.

[22] Y. Zhou et al., "Graph clustering based on structural/attribute similarities," in *VLDB'09*.

[23] B. Bollobas, *Random Graphs*. Cambridge University Press, 2001.

[24] M. E. J. Newman et al., "Random graphs with arbitrary degree distributions and their applications," *Phys. Rev. E*, vol. 64, no. 2, p. 026118.

[25] B. Pittel et al., "Sudden emergence of a giant k-core in a random graph," *J. Comb. Theory Ser. B*, vol. 67, pp. 111–151.

[26] J. I. Alvarez-hamelin et al., "Large scale networks fingerprinting and visualization using the k-core decomposition," in *NIPS'06*.

[27] S. Carmi et al., "From the cover: A model of internet topology using k-shell decomposition," *PNAS*, vol. 104, no. 27, pp. 11 150–11 154, 2007.

[28] J. Cheng et al., "Efficient core decomposition in massive networks," in *ICDE'11*.

[29] A. Clauset., "Finding local community structure in networks," *Phys. Rev. E*, vol. 72, p. 026132. 2005.

[30] J. P. Bagrow, "Evaluating local community methods in networks" *J. Stat. Mech.*, vol. 2008, p. 05001. 2008.

[31] B. Viswanath et al., "An analysis of social network-based sybil defenses," *SIGCOMM Comput. Commun. Rev.*, vol41, pp363–374.

[32] W. Cui et al., "Online Search of Overlapping Communities," in *SIGMOD '13*.