World Wide Web (2016) 19:1179–1202
DOI 10.1007/s11280-015-0378-5

# Searching overlapping communities for group query

**Jing Shan**[1] · **Derong Shen**[1] · **Tiezheng Nie**[1] · **Yue Kou**[1] · **Ge Yu**[1]

Received: 30 April 2015 / Revised: 10 November 2015 /
Accepted: 17 November 2015 / Published online: 2 December 2015

**Abstract** In most real life networks such as social networks and biology networks, a node often involves in multiple overlapping communities. Thus, overlapping community discovery has drawn a great deal of attention and there is a lot of research on it. However, most work has focused on community detection, which takes the whole network as input and derives all communities at one time. Community detection can only be used in offline analysis of networks and it is quite costly, not flexible and can not support dynamically evolving networks. Online community search which only finds overlapping communities containing a given node is a flexible and light-weight solution, and also supports dynamic graphs very well. However, in some scenarios, it requires overlapping community search for group query, which means that the input is a set of nodes instead of one single node. To solve this problem, we propose an overlapping community search framework for group query, including both exact and heuristic solutions. The heuristic solution has four strategies, some of which are adjustable and self-adaptive. We propose two parameters node degree and discovery power to trade off the efficiency and quality of the heuristic strategies, in order to make

This paper is an extended version of our previous conference paper [24].

✉ Jing Shan
mavisshan0129@gmail.com

Derong Shen
shenderong@ise.neu.edu.cn

Tiezheng Nie
nietiezheng@ise.neu.edu.cn

Yue Kou
kouyue@ise.neu.edu.cn

Ge Yu
yuge@ise.neu.edu.cn

1 College of Information Science and Engineering, Northeastern University, Shenyang, China

them satisfy different application requirements. Comprehensive experiments are conducted and demonstrate the efficiency and quality of both exact and heuristic solutions.

**Keywords** Community search · Social networks · Graph mining

## 1 Introduction

In many real-world networks such as social media and biology, community structure [13] commonly exists. We know that nodes in one community are more highly connected with each other than with the rest of the network. Thus, community structure can provide rich information of the network. For example, community in social media reflects a group of people who interact with each other more frequently, so they may have common interest or background; community in protein-association network reflects a group of proteins perform one common cellular task. Therefore, community discovery [9, 11, 12, 17] is crucial for understanding the structural and functional properties of networks.

However, communities are often not separated in most real networks, they are often overlapped. In other words, one node often belongs to multiple communities. This phenomenon could be easily explained in social media: individuals could belong to numerous communities related to their social activities, hobbies, friends and so on. Thus, overlapping community detection (OCD) [10, 14, 18, 20] has drawn a lot of attention in recent years. OCD dedicates to find all overlapping communities of the entire network, which has shortcomings in some applications: First, it is time consuming when the network is quite large. Second, OCD uses a global criterion to find communities for all nodes in a network, which is inappropriate when the density of the network distributes quite unevenly. Third, OCD can not support dynamically evolving graphs, which is a typical characteristic for most real networks especially social network. Due to these reasons, overlapping community search (OCS) problem was proposed by Cui et al. [8].

OCS finds overlapping communities that a specific node belongs to. Thus, to support online query, OCS only needs to specify the query node and discover communities locally. Hence, OCS is light-weight, flexible, and can support dynamically evolving networks. However, in some scenarios, we need to search overlapping communities for group query, that means, the input is a set of nodes instead of one single node. For example, suppose a piece of news published on social network has been read by a group of people, the service provider wants to push the news to user communities in which people will also be interested in this news; or a product has been bought by a group of customers, and the producer wants to investigate which consumer groups will also buy the product. In these scenarios, simply iterating the OCS algorithm for each query node could waste many computations and affect the efficiency. To this end, in this paper we propose an overlapping community search framework for group query (OCS-G). Because of the computation intractability of the exact solution, in order to scale up to million-node large graphs, we also proposed heuristic solution which has four strategies, some of which are adjustable and self-adaptive. Two parameters node degree and discovery power are used to trade off the efficiency and quality of the heuristic strategies, thus they could satisfy different application requirements. To the best of our knowledge, this is the first work which discusses the overlapping community search problem for group query. In summary, we make the following contributions:

- We introduce an overlapping community search framework for group query.
- We propose an efficient exact solution of overlapping community search for group query.
- We also propose a series of heuristic strategies which trade off the efficiency and quality to suit different requirements, and propose an effective adjusting parameter called "discovery power".
- We conduct comprehensive experiments on real networks to demonstrate the efficiency and effectiveness of our algorithms and theories.

The rest of this paper is organized as follows. In Section 2 we review the related work. We formalize our problem in Section 3 and propose the overlapping community search framework for group query. In Section 4 we introduce the exact solution. In Section 5 we propose four heuristic strategies. We present our experimental results in Section 6. Finally, Section 7 concludes the paper.

## 2 Related work

Our work is related to overlapping community detection problem, and local community detection problem, which can also be called community search problem.

Palla et al. first addressed overlapping community structure existing in most real networks [20], they proposed a clique percolation method (CPM), in which a community was defined as a $k$-clique component. Based on CPM, they developed a tool CFinder [1] to detect communities on biology networks. Clique percolation was successfully used on real-world networks [15, 16]. Besides structure based method like CPM, overlapping community detection could also be modeled as link-partition problem [2, 10, 18]. It first converts the original graph $G$ into link graph $L(G)$, in which each node is a link of $L(G)$, and two nodes are adjacent if the two links they represent have common node in $G$. Then link partition of $G$ can be mapped to node partition of $L(G)$, and by performing random walk [10], Jaccard-type similarity computation [2], or density-based clustering algorithm SCAN [18], node clusters of $L(G)$ are derived and then they can be converted to overlapping node communities of $G$. Label propagation method has been also widely used for detecting communities [14, 26], they propagate all nodes' labels to their neighbors for one step to make their community membership reach a consensus. Compared to OCD, OCS is more light-weight and flexible, it only needs to explore a part of the graph around the query nodes, but not the whole graph, thus it is more appropriate for online query.

Considering the scalability of community detection, local community detection problem, also called community search, has also received a lot of attention [3, 4, 6, 7, 11, 19, 22, 23]. These methods start from a seed node or node set, and then attach adjacent nodes to the community as long as these nodes can increase some community quality metrics such as local modularity [7], subgraph modularity [19], or node outwardness [3]. In [4, 23], a localized community detection algorithm is proposed based on label propagation. In [25], the community is searched in an opposite way: they take the entire graph as input and delete one node which violates the condition such as minimum degree at each step, the procedure iterates until the query nodes are no longer connected or one of the query nodes has the minimum value of the condition. Although these community search methods are more flexible than OCD, none of these methods can discovery overlapping communities, they can just find one community or separated communities.

Our work is inspired by Cui et al. [8], they proposed online overlapping community search problem. They defined a community as a $k$-clique component, and an algorithm which finds overlapping communities a given node belongs to was given. However, the algorithm of OCS still has a large room for performance improvement, and also, they did not consider the problem of overlapping community search for group query. Although simply iterating the algorithm in [8] could solve the problem, this method could produce a lot of waste computations and it is not an effective solution. Thus, we propose an OCS framework for group query, including both exact and heuristic solutions. As far as we know, this is the first work considering OCS problem for group query. Besides, in [8], they just proposed an unadjustable approximate algorithm, the efficiency and result quality of this algorithm can not be controlled. But some of our heuristic strategies are adjustable and self-adaptive. By tuning the adjusting parameters, we could trade off the efficiency and result quality, and make them to satisfy different requirements.

The differences of this extended manuscript from our conference version [24] are as follows:

– We focused on the problem of searching overlapping communities for group query, and reorganized the structure of the paper to make the expression clearer.
– We proposed an adjust parameter called discovery power, and utilized it in the heuristic algorithm, Section 5.3 was newly added.
– We proposed an advanced heuristic algorithm, which was self-adaptive and utilized discovery power to adjust the efficiency and quality of the heuristic algorithm, and Section 5.4 was newly added.
– We conducted experiments on a new added dataset called Friendster, and compared the efficiency of the four heuristic strategies, and also added the evaluation of the efficiency and quality performance of the advanced heuristic strategy. Besides, we evaluated the influence of the adjust parameter discovery power.

## 3 Problem definition and framework

In this section, we define the problem of overlapping community search for group query more formally, and propose the OCS framework for group query.

### 3.1 Problem definition

Intuitively, a typical member of a community is linked with many but not necessarily all other nodes in the community. Base on this basic principle, there are many types of definitions as mentioned in [21], which could be classified into three classes: local definitions, global definitions, and process-based definitions. Local definitions consider the community-ness of vertices locally, which means detecting communities on a connected subset of vertices, such as $n$-clubs, $n$-clans, $k$-cores and so on; global definitions consider the community-ness of vertices globally, which means detecting communities on the whole

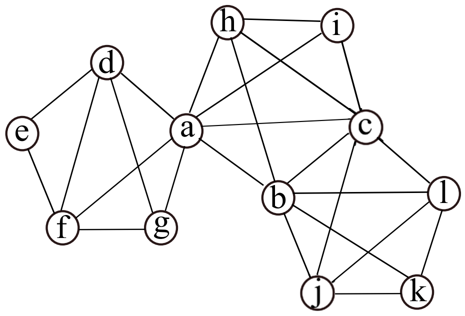
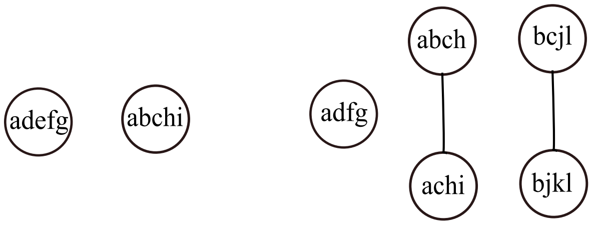


**Figure 1** A toy social network graph

network, such as normalized cut, conductance, modularity and so on; process-based definitions define community by means of considering community formation process taking place on the network under study, the most typical definition is the Clique Percolation Method (CPM) proposed by Palla et al. [20], this method considers a $k$-clique template rolling on the network and resulting in a community. This community formation process based definition is the most suitable for the idea of searching a community from a vertex on the graph, and it could find overlapping communities. So we use $k$-clique as building blocks of a community just as CPM: given a graph, we can derive a *k-clique graph* in which each node represents a $k$-clique, and if two cliques share $k-1$ nodes, there exists an edge between them. A *community* is defined as a $k$-clique component, which is a union of all $k$-cliques sharing $k-1$ nodes.

However, the definition of community above is too strict. Therefore, Cui et al. [8] proposed a less strict definition: two $k$-cliques are adjacent if they share at least $\alpha$ nodes, where $\alpha \leq k-1$; and $k$-clique can be replaced by $\gamma$-quasi-k-clique [5] in which $k$ nodes have at least $\lceil \gamma \frac{k(k-1)}{2} \rceil$ edges. Now, we give the problem definitions of OCS for single query node and group query.

**Problem 1** (($\alpha, \gamma$)-OCS) For a graph $G$, a query node $v_0$ and a positive integer $k$, the ($\alpha, \gamma$)-OCS problem finds all $\gamma$-quasi-$k$-clique components containing $v_0$, and two $\gamma$-quasi-$k$-clique nodes of one component are adjacent if they share at least $\alpha$ nodes, where $\alpha \leq k-1$.



(a) The clique graph for $(4, 0.8)$-OCS
(b) The clique graph for $(3, 1)$-OCS-G

**Figure 2** Clique graphs for OCS-G

*Example 1* For the graph shown in Figure 1, suppose we want to find all the communities which contain node $a$ ($v_0$ = a), let $k = 5$, $\alpha = 4$, $\gamma = 0.8$, we have two 0.8-quasi-5-cliques containing $a$: $C_1$ = adefg, $C_2$ = abchi, the clique graph is shown in Figure 2a. Thus, we get two communities $\{a, d, e, f, g\}$ and $\{a, b, c, h, i\}$ containing $a$.

**Problem 2** (($\alpha$, $\gamma$)-OCS-G) For a graph $G$, a group query $V_q$ which is a node set and a positive integer $k$, the ($\alpha$, $\gamma$)-OCS-G problem finds all $\gamma$-quasi-$k$-clique components containing at least one node in $V_q$, and two $\gamma$-quasi-$k$-clique nodes of one component are adjacent if they share at least $\alpha$ nodes, where $\alpha \leq k - 1$.

*Example 2* For the graph in Figure 1, suppose given a group query $V_q$ = {a,b,c}, let $k$ = 4, consider (3, 1)-OCS-G, as shown in Figure 2b we get three communities $\{a, d, g, f\}$, $\{a, b, c, h, i\}$, $\{b, c, j, k, l\}$ containing nodes in $V_q$.

Apparently, both OCS problem and OCS-G problem are NP-hard, because they can be reduced from $k$-clique problem.

## 3.2 Framework of OCS-G

As shown in Algorithm 1, when given a group query, for each node $v_i$ in $V_q$, we first find a clique containing $v_i$, and then find the clique component which the clique belongs to. Notice that for a clique component (i.e. a community), we derived the same component no matter which clique of it is started from. Thus, to avoid redundant enumeration, we only enumerate unvisited cliques for each round of iteration. Base on this framework, we propose both exact and heuristic solutions for OCS-G, and the details are introduced in the following two sections, note that our solutions are not just iterating the OCS algorithm for each node.

**Algorithm 1** Framework of OCS-G

```
Input: $G(V, E)$, $V_q$, $\alpha$, $\gamma$, $k$;
Output: The overlapping communities containing $\forall v_i \in V_q$
1 $\mathcal{R} \leftarrow \emptyset$;
2 foreach $v_i \in V_q$ do
3     while $C \leftarrow next\_clique(v_i), C \neq \emptyset$ do
4         $\mathbf{C} \leftarrow expand(C)$;
          // find the clique component $\mathbf{C}$ of $C$
5         $\mathcal{R} \leftarrow \mathcal{R} \cup \mathbf{C}$;
6 Return $\mathcal{R}$;
```

# 4 Exact solution of OCS-G

In this section, we first optimize the OCS algorithm by boundary node limitation, then we give the exact solution of OCS-G, both of which are based on a series of theorems with strong proofs.

## 4.1 Optimized OCS algorithm

OCS algorithm searches overlapping communities of one single input node. For a query node $v$, OCS first finds a clique containing $v$ by $next\_clique()$, and then finds the clique component which the clique belongs to by $expand()$. We omit the details and refer readers who are interested to [8]. The function and basic idea of $next\_clique()$ and $expand()$ are as follows:

- $next\_clique()$: It is in charge of enumerating each $\gamma$-quasi-$k$-clique containing $v$. It starts from $v$, and when traverses to a new adjacent node of the current node, iteratively takes the adjacent node as start node until a new valid clique is found. The iterative procedure will stop until all valid cliques contain $v$ are found.
- $expand()$: It is used to find the clique component of $Clique\ C$. It starts from $C$, and we replace $C$'s each subset $S_1$ with $S_2$, in which each node is the neighborhood of $C - S_1$, and $|S_1| = |S_2| \leq k - \alpha$. If the new combination $(C - S_1) \cup S_2$ is a valid clique, we iteratively take the new clique as start clique. The iterative procedure will stop until all cliques of the clique component are found.

Though the DFS procedures of $next\_clique()$ and $expand()$ are pruned by checking the edge number of subgraph induced by current visiting node set in [8], we find a way which could further prune nodes to be checked. To optimize OCS algorithm, we first introduce two definitions, interior node and boundary node.

**Definition 1** (Interior Node) In the process of searching community $\mathbf{C_m}$, given a node $i$, if $i$ and all its neighbors $neighbor(i)$ both exist in the currently found result set of $\mathbf{C_m}$, we say $i$ is an interior node.

**Definition 2** (Boundary Node) In the process of searching community $\mathbf{C_m}$, given a node $b$, if $b$ exists in the result set of $\mathbf{C_m}$, and one or more neighbors of $b$ do not exist in the result set, we say $b$ is a boundary node.

Nodes which are not interior nodes or boundary nodes are called exterior nodes. Based on the three types of nodes, we propose a theorem to optimize the OCS algorithm.

**Theorem 1** *For OCS algorithm, one community can be derived by only expanding boundary nodes and exterior nodes without losing completeness.*

*Proof* At the beginning of searching community $\mathbf{C_m}$, every node is exterior node, thus a clique containing the query node can be found. In the procedure of expanding the clique, suppose set $R$ is the result set including nodes of $\mathbf{C_m}$ that have already been found, node $i$ is an interior node, if there exists an exterior node $n \in \mathbf{C_m} - R$, and it can be added into $R$ from $i$, there must exist a clique $C_l$ including $i$ and $n$. Because $n$ must be connected to at least one node $b$ in $C_l - n$, thus the node $b$ is a boundary node, and $n$ can be added into $\mathbf{C_m}$ from $b$, therefore the theorem holds. □

According to Theorem 1, when expanding the current clique, if candidate nodes which are used to replace the current clique nodes are interior nodes, these nodes can be skipped. Besides, node degree could be taken into consideration as a pruning condition. For a $\gamma$-quasi-$k$-clique, the minimum degree of a node should be $d_{min}(v) = \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$. Based on the definition of community, if one node has less than $d_{min}(v) = \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$ edges, it is impossible to belong to a community. When $\gamma = 1$, the node should have at least $k - 1$ edges. Thus, utilizing interior node and node degree as pruning conditions, we could optimize *next_clique*() and *expand*() of OCS as depicted in Algorithms 2 and 3.

**Algorithm 2** optimized next_clique($v_0$)

```
   Input: $v_0$: a query node
   Output: $C$: next $\gamma$-quasi-$k$-clique
1  $U \leftarrow \{v_0\}$;
2  DFS ($U, v_0$);
3  Procedure DFS($U, u$)
4      if $|U| = k$ then
5          if $U$ is a $\gamma$-quasi-$k$-clique and $U$ is unvisited then
6              return $U$;
7          else
8              return;
9      if $g(U) < \gamma \frac{k(k-1)}{2}$ then
10         return;
11     foreach $(u, v) \in E, v \notin U$ do
12         if neighbor$(v) \geq \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$ then
13             DFS($U \cup \{v\}, v$)
```

As mentioned before, we use DFS strategy to traverse nodes from the query node. Traversed nodes are iteratively added into set $U$, and check if a new valid clique is found (Algorithm 2 line 4-8), we use node degree condition to prune nodes needed to be traversed (line 12), $neighbor(v)$ represents the degree of $v$. Besides, $g(U)$ is another pruning condition proposed in the original OCS Algorithm [8](line 9), it represents the maximal number of edges that the result clique has, and

$$g(U) = |E(U)| + (k - |U|)|U| + \frac{(k - |U|)(k - |U| - 1)}{2}, \tag{1}$$

where $|E(U)|$ is the number of edges in the subgraph induced by $U$.

**Algorithm 3** optimized expand(C)

```
   Input: $C$: a $\gamma$-quasi-$k$-clique
   Output: $A$: the community of $C$
1  $A \leftarrow C$;
2  EXPAND_CLIQUE ($C$);
3  return $A$;
4  Procedure EXPAND_CLIQUE($C$)
5      sort $C$ by $d_{nc}(n)$, $n \in C$;
6      foreach $S_1 \in C$ and $|S_1| \geq \alpha$ do
7          $S_2 = C - S_1$;
8          foreach $u \in S_1$ do
9              foreach $v \in neighbor(u)$ do
10                 if $d_{nc}(v) > 0$ and $neighbor(v) \geq \lceil \gamma\binom{k}{2} - \binom{k-1}{2} \rceil$ then
11                     $Cand \leftarrow Cand \cup v$;
12         if $|Cand| \leq |S_2|$ or $g(S_1) < \gamma\frac{k(k-1)}{2}$ then
13             Continue;
14         foreach $S_2' \in Cand, |S_2| = |S_2'|$ do
15             $C' \leftarrow S_1 \cup S_2'$;
16             if $C'$ is unvisited and $C'$ is a $\gamma$-quasi-$k$-clique then
17                 $A \leftarrow A \cup S_2'$;
18                 Update($A, S_2'$);
19                 EXPAND_CLIQUE($C'$);
```

After find a clique $C$, we use $expand(C)$ to get the clique component of $C$, which can constitute a community. We adopt a DFS traversal on the clique graph. The key operation of the expanding procedure is to replace subset $S_2$ of $C$ ($|S_2| \leq k-\alpha$) with the remaining subset $S_1$'s ($|S_1| \geq \alpha$) neighbors $S_2'$ (line 15), where $|S_2'| = |S_2|$, and these neighbors should satisfy 1) they are not interior nodes, 2) their degree should be not less than the lower bound (line 10). We use $d_{nc}(v)$ to denote the number of $v$'s neighbors which are not in a community, $d_{nc}(v) = 0$ means $v$ is an interior node of the current explored community. Note that $d_{nc}$ is defined on the nodes which are already in the current community result set, if node $v$ is not in the result set, its $d_{nc}(v)$ is unknown, and we initialize the value of $d_{nc}(v)$ with the degree of the node at the beginning. For the new combination $C'$, we check if it is a new valid clique (line 16), if so, $S_2'$ is added into the result set $A$ (line 17) and $d_{nc}$ values of nodes in A need to be updated (line 18), then we expand $C'$ (line 19). Note that at the beginning of expand procedure, we sort nodes of clique $C$ by $d_{nc}$ in ascending order (line 5), then we pick nodes of $C$ by the order to form $S_1$. By doing this, we could guarantee that nodes with lower $d_{nc}$ value change into interior nodes earlier, and we could get more interior nodes as early as possible.

Benefited from interior node and node degree pruning conditions, the enumerations of finding and expanding clique are sharply reduced. Thus the efficiency of OCS algorithm is highly improved, and this is shown by experiments in Section 6.

### 4.2 Exact OCS-G solution

After optimizing OCS algorithm, we could utilize it for each node in the group query for OCS-G problem. However, when it comes to OCS-G problem, there is still room for efficiency improvement. Instead of simply iterating OCS, we try to avoid repeated computations by utilizing existing results. Note that there exists a consistency property for OCS problem:

*Property 1* In $(\alpha, \gamma)$-OCS, if $\mathbf{C_m}$ is a community that contains query node $v_0$, for any other node $v \in \mathbf{C_m}$ as query node, $\mathbf{C_m}$ is also returned as its community.

According to Property 1, we can get a corollary to save some duplicate steps which involve in query nodes.

**Corollary 1** *Given a group query $V_q = \{v_1, ...v_n\}$, we do OCS-G and take input nodes in the order of their subscripts, when taking $v_i$ as input node, if any node in $\{v_1, ..., v_{i-1}\}$ can be traversed, it can be skipped.*

*Proof* Suppose $v_j$ $(1 \leq j \leq i-1)$ is traversed when input node is $v_i$, due to the completeness of OCS algorithm, all communities containing $v_j$ have been found when taking $v_j$ as input node, and we also have that all communities containing $v_i$ and $v_j$ are also contained in communities containing $v_j$, so all communities containing $v_i$ and $v_j$ also have already been found, thus the traversal can skip $v_j$. □

Consider Example 2, suppose we already finished the first round taking $a$ as input node and got $Community\{a, d, f, g\}$, $\{a, b, c, i, h\}$, and now consider node $b$ as input. Intuitively, since we already got $\{a, b, c, i, h\}$, according to Property 1, when we take $b$ as input node, we will still get $\{a, b, c, i, h\}$. Thus, we wonder if we could omit some traversals related to $\{a, b, c, i, h\}$. According to Corollary 1, query node $a$ can be skipped. The ideal situation is that all nodes in $\{a, b, c, i, h\}$ could be skipped, however, if we do that, we could only get $\{b, l, j, k\}$ as the result of the second round, and the exact result should be $\{b, c, l, j, k\}$. Apparently, node $c$ is missing. So we try to find besides query nodes, which nodes in the existing community can be skipped and which can not, and we get Theorem 2.

**Theorem 2** *For $(k-1, 1)$-OCS-G, given a node $v$ which is a member of existing community $\mathbf{C_m}$, and the node $v$'s degree $d(v) \leq k$, then $v$ cannot exist in a new community $\mathbf{C'_m}$.*

*Proof* Suppose $v \in \mathbf{C'_m}$, so there exists a clique $C'_l$: $vn'_1 \ldots n'_{k-1}$ which belongs to $\mathbf{C'_m}$, and the degree of $v$ in $C'_l$ is $d_{C'_l}(v) = |n'_1 \ldots n'_{k-1}| = k-1$, and we know that $v \in \mathbf{C_m}$, so there exists a clique $C_l$: $vn_1 \ldots n_{k-1}$ which belongs to $\mathbf{C_m}$ and $d_{C_l}(v) = |n_1 \ldots n_{k-1}|$ $= k-1$. Because $C_l$ and $C'_l$ are not in the same community, they are not adjacent, and satisfy $|C_l \cap C'_l| < k-1$, so we have $|(C_l - v) \cap (C'_l - v)| < k-2$. We know that $d_{min}(v) = |(C_l - v) \cup (C'_l - v)| = |C_l - v| + |C'_l - v| - |(C_l - v) \cap (C'_l - v)|$, so by computation we can derive $d(v) > k$, and this conflicts with the condition $d(v) \leq k$. Therefore, the theorem holds. □

According to Theorem 2, we could easily infer that for $(k-1, 1)$-OCS-G, if a node already exists in a community and its degree is not larger than k, it can be skipped when traversed. Consider the example above, only $d(c)$ is larger than 4, it cannot be skipped, other nodes $a, h, i$ can be skipped during DFS procedure taking $b$ as input in the second round.

Now we discuss which nodes can be skipped for $(\alpha, \gamma)$-OCS-G. For a $\gamma$-quasi-$k$-clique, the minimum degree of a node should be $d_{min}(v) = \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$, and to keep the clique connected, $d_{min}(v) \geq 1$. Also, if two quasi cliques are not in the same community, they share less than $\alpha$ nodes. Thus, we replace the conditions in the proof of Theorem 2 and get Theorem 3

**Theorem 3** *For $(\alpha, \gamma)$-OCS-G, given a node $v$ which is a member of existing community $\mathbf{C_m}$, and the node $v$'s degree $d(v) \leq max\{2\lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil - (\alpha - 1),\ \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil\}$, then $v$ cannot exist in a new community $\mathbf{C'_m}$.*

*Example 3* For the graph in Figure 1, suppose a group query $V_q = \{a, b, c\}$, let k = 5, consider $(3, 0.9)$-OCS-G, after the first round of input node $a$, we get $Community\{a, b, c, h, i\}$, when taking input node $b$ in the second round, according to Theorem 3, we only need to traverse nodes with $d(v) > 4$, thus $h$ and $i$ can be skipped during the DFS procedures of $next\_clique()$ and $expand()$.

From Example 3 we can see that utilizing Theorem 3, the performance of OCS-G Algorithm is remarkably improved. However, taking $(k-1, 1)$-OCS-G as example, the lower bound of community node degree is $k$, which is not big enough for efficient pruning. Thus, we further discover other pruning rules. Based on the definitions of interior and boundary node, we have Theorem 4:

**Theorem 4** *If one node $i$ is an interior node of existing community $\mathbf{C_m}$, it cannot exist in a new community $\mathbf{C'_m}$*

*Proof* We know that node $i$ exists in community $\mathbf{C_m}$, suppose it still exists in community $\mathbf{C'_m}$, then there exists a clique $C'_l$: $in'_1 \ldots n'_{k-1}$ which belongs to $\mathbf{C'_m}$, because community $\mathbf{C_m} \neq \mathbf{C'_m}$, thus there exists at least one node of $n'_1 \ldots n'_{k-1}$ which is not in community $\mathbf{C_m}$, this conflicts with that node $i$ is an interior node of $\mathbf{C_m}$, therefore the theorem holds. □

**Algorithm 4** modify next_clique($v_0$)

```
1 if $d_{nc}(v) > 0$ then
2     if $v \in R$ then
3         if $neighbor(v) > max\{2\lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil - (\alpha - 1),\ \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil\}$ then
4             DFS($U \cup v, v$);
5     else
6         if $neighbor(v) \geq \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$ then
7             DFS($U \cup v, v$);
```

According to Theorem 4, after get the first community by expanding a clique, we could find the next clique of a query node by only traversing boundary nodes and exterior nodes in OCS-G problem. Utilizing Theorems 3 and 4, we could modify Algorithm 4 by replacing line 12-13 with Algorithm 4. When traversing to a node $v$, we first check if it is not an interior node (line 1), then check if it is already in an existing community (line 2), $R$ represents the result set of OCS-G, if the node already exists in a community, we use the lower bound mentioned in Theorem 3 as pruning condition (line 3); if it does not exist in a community, we use the lower bound of node degree mentioned in Algorithm 2 (line 6).

Similarly, Algorithm 3 could also be modified in OCS-G problem, we could replace line 10-11 with Algorithm 4, in which line 4 and line 7 are changed into $Cand \leftarrow Cand \cup v$.

After modify Algorithms 2 and 3, we apply them in the OCS-G framework which is shown in Algorithm 1, thus we get the exact solution of OCS-G, the improvement of performance will be shown through experiments in Section 6.

**Complexity analysis** Consider a $k$-clique community $\mathbf{C_m}$, the time complexity of the exact OCS-G solution is $O(\binom{|\mathbf{C_m}|}{k})$. This is the worst time cost when all the nodes in $\mathbf{C_m}$ are boundary nodes, and other filtering conditions can not be satisfied either. But when there exist a certain amount of interior nodes, the performance of our optimized exact algorithm is greatly improved. Since our algorithm does not utilize any index, the space complexity is $O(|\mathbf{C_m}|)$.

## 5 Heuristic strategies of OCS-G

Although the performance of the exact solution has been greatly improved, it is still an NP-hard problem. Thus, to scale up to million-node large graphs, we propose a series of heuristic strategies for OCS-G problem. First we give a naive heuristic strategy which discards all boundary nodes and only traverses exterior nodes. Second we propose two adjustable heuristic strategies intuitively using node degree to adjust the proportion of nodes needed to be explored. Then we introduce an advanced parameter discovery power which could better adjust the efficiency and quality of the heuristic strategy. At last we present an advanced adjustable heuristic strategy which utilizes discovery power to control the computing process.

### 5.1 Naive heuristic strategy

Considering the search process of one community $\mathbf{C_m}$, it starts from the query node, and adjacent nodes are added into $\mathbf{C_m}$ as long as they satisfy that 1) they belong to a $\gamma$-quasi-$k$ clique, 2) the clique they belong to can be reached from the start clique, 3) they are not in $\mathbf{C_m}$.

We see the community as a growing circle with nodes scattering in it, if we explore the nodes out of the circle, but not wander in the circle, the entire community can be found more earlier. According to Theorems 1 and 4, the exact solution only traverse boundary nodes and exterior nodes. Intuitively, the naive heuristic strategy can omit all boundary nodes and just traverse the exterior nodes, Thus we have:

**Strategy 1** (Naive Heuristic Strategy) *Traverse only exterior nodes during* $expand()$ *procedure. That is, skip interior and boundary nodes when expanding the current community.*

Note that we only apply our heuristic strategy on $expand()$ procedure, $next\_clique()$ procedure is still exact. Because compared to expanding seed cliques, the computations of finding a new clique as seed occupy only a small portion of the whole procedure. But if we lose one seed clique, we may miss a bunch of cliques which could be reached from the seed clique. Thus, to guarantee the quality of the results, we only apply it on $expand()$ procedure.

## 5.2 Adjustable heuristic strategies

Strategy 1 only use boundary node as filter condition to prune nodes to be traversed. However, boundary node is an uncontrollable condition, which is determined by the nature characteristic of the graph. Thus, we introduce a parameter which could adjust the computing procedure.

Intuitively, nodes with higher degree have more possibility to belong to one or more cliques. Thus, if we want to prune traversed nodes during the process, we could raise the lower bound of $neighbor(v)$, the higher the lower bound is, the less traversed nodes are, and the more the quality loss is. Combined node degree restriction with boundary node, we could form two adjustable heuristic strategies:

**Strategy 2** (Adjustable Heuristic Strategy (fine)) *Traverse boundary nodes with node degree restriction and all exterior nodes during $expand()$ procedure. That means partially traversing boundary nodes and completely traversing exterior nodes.*

**Strategy 3** (Adjustable Heuristic Strategy (coarse)) *Traverse only exterior nodes with node degree restriction during $expand()$ procedure. That means skipping all boundary nodes, and partially traversing exterior nodes.*

Theoretically, the relation of efficiency and quality of these three strategies is as depicted in Proposition 1, and the proof of the proposition is obvious so we omit it for brevity.

**Proposition 1** *For the three heuristic strategies of OCS-G, the relation of their efficiency is $Strategy\ 2¡Strategy\ 1¡Strategy\ 3$, and the relation of their quality is $Strategy\ 2 > Strategry\ 1 > Strategy\ 3$.*

However, using node degree as the adjusting parameter is an intuitive method, and this method is limited. For example, if there exists a node whose degree is very large, but its neighbors can form few cliques with it. In this situation, it will waste the searching cost and influence the accuracy. Thus we try to seek a reasonable parameter to tune the efficiency and quality, and we call it “discovery power”.

## 5.3 Discovery power

For heuristic strategies, as the efficiency improves, the result quality will be sacrificed, thus the adjusting parameter could be used to tune the efficiency and quality to make it satisfy different requirements. So we try to seek a reasonable adjusting parameter with strong theoretical support. To this end, we introduce $E(N_v^k)$, which is the expectation of $k$-clique number that node $v$ belongs to.

Now we introduce the method of computing $E(N_v^k)$. Suppose node $v$ has $d(v)$ neighbors, and there exist $|e|$ edges between $v$'s neighbors, the problem of calculating $E(N_v^k)$ can be transformed to calculating $E(N^{k-1}_{Nb(v)})$, which is the expectation of the number of $(k-1)$-cliques existing in $v$'s neighbors $Nb(v)$. The transformation is reasonable because each node in $Nb(v)$ is connected to node $v$, thus if there exists a $(k-1)$-clique in $Nb(v)$, this $(k-1)$-clique and $v$ can form a $k$-clique.

Considering $E(N^{k-1}_{Nb(v)})$, given $d(v)$ nodes and $|e|$ edges, there are $p = \binom{\binom{d(v)}{2}}{|e|}$ ways to place $|e|$ edges, and each way is equally likely with the probability $1/p$. For each placing solution, suppose the corresponding value of the random variable $N^{k-1}_{Nb(v)}$ is $n_i$, then the expectation can be defined as:

$$E(N^{k-1}_{N(b)}) = \frac{1}{p}\sum_{i=1}^{p} n_i \ , \tag{2}$$

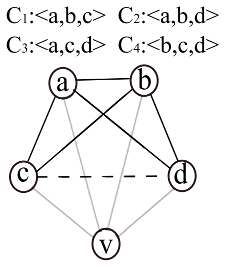
where $p = \binom{\binom{d(v)}{2}}{|e|}$.

Now we discuss the semantics of the summation $\sum_{i=1}^{p} n_i$, it represents the clique number summation of all edge placing solutions. Note that besides adding up the clique number of each solution, we could compute the summation from another perspective: transform the summation of clique number to the summation of clique count, which is the number of times that a clique occurs in all the solutions.

Given $d(v)$ nodes and $|e|$ edges, there are $\binom{d(v)}{k-1}$ ways to choose a $(k-1)$-clique. Once the clique is chosen, the way of placing the clique's edges is set. For each chosen clique, there are$\binom{\binom{d(v)}{2}-\binom{k-1}{2}}{|e|-\binom{k-1}{2}}$ ways to place the remaining edges. Thus, the summation can be defined as:

$$\sum_{i=1}^{p} n_i = \binom{d(v)}{k-1}\binom{\binom{d(v)}{2}-\binom{k-1}{2}}{|e|-\binom{k-1}{2}} \tag{3}$$

*Example 4* Consider the graph shown in Figure 3, node $v$ has 4 neighbors ($d(v) = 4$), suppose there exist 5 edges between these nodes ($|e| = 5$), to compute the summation of 3-clique number ($k-1=3$) of all edge placing solutions, we can see that there are 6 ways of placing these edges, and the clique number of each way is 2, thus the clique number summation $\sum_{i=1}^{p} n_i = 12$. The tables show the transformation, we can see that the clique count of each clique is 3, thus the clique count summation is also 12. And $\sum_{i=1}^{p} n_i = \binom{4}{3}\binom{\binom{4}{2}-\binom{3}{2}}{5-\binom{3}{2}} = 12$, which shows the correctness of (3).



| Solution | Clique ID |
|---|---|
| 1 | C1, C2 |
| 2 | C1, C3 |
| 3 | C1, C4 |
| 4 | C2, C3 |
| 5 | C2, C4 |
| 6 | C3, C4 |

| Clique ID | Solution |
|---|---|
| C1 | 1, 2, 3 |
| C2 | 1, 4, 5 |
| C3 | 2, 4, 6 |
| C4 | 3, 5, 6 |

**Figure 3** A example of computing $\sum_{i=1}^{p} n_i$ with $d(v) = 4$, $|e| = 5$, $k-1=3$

Finally, we get the expression of $E(N_v^k)$ as shown in (4), which is derived from (2).

$$E(N_v^k) = E(N_{Nb(v)}^{k-1}) = \frac{\binom{d(v)}{k-1}\binom{\binom{d(v)}{2}-\binom{k-1}{2}}{|e|-\binom{k-1}{2}}}{\binom{\binom{d(v)}{2}}{|e|}}, \tag{4}$$

where $Nb(v)$ is a node set containing all the neighbors of node $v$, and $|Nb(v)| = d(v)$.

Consider the graph shown in Figure 3, given node $v$, $d(v) = 4$, $|e| = 5$, to compute the expectation of 4-clique number of $v$, we have $E(N_v^k) = 12/\binom{\binom{4}{2}}{5} = 2$.

Note that (4) computes the expectation of $k$-clique number that node $v$ belongs to, when considering $(\alpha, \gamma)$-OCS-G problem, the discovery power parameter should be changed to $E(N_v^{\gamma k})$, which is the expectation of $\gamma$-quasi-$k$-clique number that node $v$ belongs to. However, when node $v$ belongs to a $\gamma$-quasi-$k$-clique, the minimum degree value of node $v$ is $d_{min}(v) = \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$, which means that node $v$ does not need to be connected to all the other $k-1$ nodes in the quasi-clique. Thus, given a node $v$, even nodes which are not $v$'s neighbors can form a quasi-clique with $v$. Because of this reason, to directly compute $E(N_v^{\gamma k})$ is difficult. So for $\gamma$-quasi-$k$-clique, we reduce $k$ to $d_{min}(v) + 1$, and the discover power of $(\alpha, \gamma)$-OCS-G problem is defined as $E(N_v^{d_{min}(v)+1})$, as shown in (5):

$$E(N_v^{d_{min}(v)+1}) = E(N_{Nb(v)}^{d_{min}(v)}) = \frac{\binom{d(v)}{d_{min}(v)}\binom{\binom{d(v)}{2}-\binom{d_{min}(v)}{2}}{|e|-\binom{d_{min}(v)}{2}}}{\binom{\binom{d(v)}{2}}{|e|}}, \tag{5}$$

where $d_{min}(v) = \lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$.

Although the parameter $k$ in discovery power of $\gamma$-quasi-$k$-clique is reduced to $d_{min}(v)+1$ and it is not the exact expectation of $\gamma$-quasi-$k$-clique, it still can reflect the possibility a node belonging to a $\gamma$-quasi-$k$-clique. Consider the graph in Figure 3, given $k = 4$, Table 1 shows the expectation of each node with $\gamma = 1$ and $\gamma = 0.8$ respectively. We can see that they have the same varying tendency.

Note that we compute the discovery power of each node during the offline phase when constructing the network, and see the parameter as a basic attribute of one node just as node degree, thus when we do OCS-G, we can use the parameter directly without any time cost.

## 5.4 Advanced adjustable heuristic strategy

After introducing the discovery power parameter, we now propose the advanced adjustable heuristic algorithm, which utilizes the parameter discovery power to adjust the efficiency and quality of the heuristic algorithm. Note that we still keep the *next_clique*() procedure exact, and only apply the heuristic strategy to the *expand*() procedure. The heuristic algorithm of *expand*() procedure is shown in Algorithm 5.

**Table 1** Expectations of nodes in Figure 3 with $k = 4$, $\gamma = 1$ and $\gamma = 0.8$

| | $E(N_a)$ | $E(N_b)$ | $E(N_c)$ | $E(N_d)$ | $E(N_v)$ |
|---|---|---|---|---|---|
| $\gamma = 1$ | 2 | 2 | 1 | 1 | 2 |
| $\gamma = 0.8$ | 5 | 5 | 3 | 3 | 5 |

**Algorithm 5** self-adapting heuristic expand(C)

```
   Input: $C$: a $\gamma$-quasi-$k$-clique
   Output: $A$: the community of $C$
1  $A \leftarrow C$;
2  EXPAND_CLIQUE $(C, 1)$;
3  return $A$;
4  Procedure EXPAND_CLIQUE$(C, l)$
5      $\eta = \frac{1}{5} l \ln l + 1$;
6      foreach $n \in C$ do
7          if $E(N_n^{d_{min}(n)+1}) \geq \eta\tau$ then
8              $C_e \leftarrow C_e \cup n$;
9      if $|C_e| < \alpha$ then
10         return;
11     else
12         sort $C_e$ by $d_{nc}(n)$, $n \in C_e$;
13         foreach $S_1 \in C_e$ and $|S_1| \geq \alpha$ do
14             $S_2 = C_e - S_1$;
15             foreach $u \in S_1$ do
16                 foreach $v \in neighbor(u)$ do
17                     if $d_{nc}(v) > 0$ then
18                         if $v \in R$ then
19                             if $neighbor(v) >$
                               $max\{2\lceil\gamma\binom{k}{2} - \binom{k-1}{2}\rceil - (\alpha - 1), \lceil\gamma\binom{k}{2} - \binom{k-1}{2}\rceil\}$ then
20                                 $Cand \leftarrow Cand \cup v$;
21                         else
22                             if $neighbor(v) \geq \lceil\gamma\binom{k}{2} - \binom{k-1}{2}\rceil$ then
23                                 $Cand \leftarrow Cand \cup v$;
24             if $|Cand| \leq |S_2|$ or $g(S_1) < \gamma\frac{k(k-1)}{2}$ then
25                 Continue;
26             foreach $S_2' \in Cand, |S_2| = |S_2'|$ do
27                 $C' \leftarrow S_1 \cup S_2'$;
28                 if $C'$ is unvisited and $C'$ is a $\gamma$-quasi-$k$-clique then
29                     $A \leftarrow A \cup S_2'$;
30                     Update$(A, S_2')$;
31                     EXPAND_CLIQUE$(C', l + 1)$;
```

As shown in Algorithm 5, we use discovery power to filter nodes in the current clique. For the nodes in the current clique, if their discovery power exceeds the threshold $\eta\tau$, we keep them in the set $C_e$ to explore their neighbor nodes (line 6-8), and if the node number in set $C_e$ is less than $\alpha$, that means it has little possibility to find a new clique which shares $\alpha$ nodes with the current clique, thus there is no new node can be added into the current community, so we give up to expand the current clique (line 9–10). If $|C_e| \geq \alpha$, we explore the neighbor nodes of $C_e$ the same way as we do in the exact algorithm (line 11–31).

Notice that Algorithm 5 is a self-adapting algorithm correlated with the discovery depth, with $\eta$ being the self-adapting parameter and $l$ representing the discovery depth (line 5), and $\tau$ being the tuning parameter which could adjust the performance and quality of the heuristic algorithm. At the beginning of the expanding procedure, the discovery depth is layer 1 (line 2), and $\eta = 1$, thus the filtering threshold is $\tau$. As the depth increases (line 31), the self-adapting parameter $\eta$ increases, thus the threshold also increases. So, as the procedure goes deeper, more nodes are filtered. The reason of doing self-adapting correlated with discovery depth is that, nodes close to the start node are more important for getting the result. If a clique is missed at the first layer, that means a group of nodes which belong to the community will be pruned. As the depth increases, the importance decay decreases. For example, the loss of a clique at the first layer is more influential than that at the second layer. However, the difference of losing a clique at the 11th layer and the 12th layer is not that much. Hence, we adopt $\eta$ as the self-adapting parameter to further balance the efficiency and the result quality of the heuristic algorithm. We will further discuss the parameter value setting of Algorithm 5 in Section 6. The advanced adjustable heuristic strategy is:

**Strategy 4** (Advanced Adjustable Heuristic Strategy) *Traverse boundary nodes and exterior nodes from expandable nodes in the current clique, and the expandable nodes' discovery power should exceed threashold $\eta\tau$. That means, only explore the nodes from the paths that most probably bring in new nodes belonging to the community.*

### 5.5 Analysis

Consider a $k$-clique community $\mathbf{C_m}$, the time and space complexity analyses of the four heuristic strategies are as follows:

– For Strategy 1, each time when a clique is visited, a new node of the current community is found. Thus, the time complexity of Strategy 1 is $O(|\mathbf{C_m}|)$, we reduce the exponential complexity to linear.
– For Strategy 2, suppose the number of boundary nodes which exceed the lower bound of node degree is $n$, notice that $n \leq |\mathbf{C_m}|$, when the lower bound increases, $n$ will decrease. The time complexity of Strategy 2 is $O(\binom{n}{k} + |\mathbf{C_m}|)$ .
– For Strategy 3, suppose the number of exterior nodes which exceed the lower bound of node degree is $n$, then the time complexity is $O(n)$.
– For Strategy 4, suppose the number of nodes whose discovery power exceeds the threshold $\eta\tau$ is $n$, the time complexity of Strategy 4 is $O(\binom{n}{k})$.
– The space complexity of the four heuristic strategies is $O(|\mathbf{C_m}|)$, because no index is utilized in these strategies.

Due to the *NP-hardness to approximate* of clique finding, it is difficult to give theoretic guarantee of the four heuristic strategies. We could only evaluate the quality through experiments in Section 6.

## 6 Experimental study

In this section, we present experimental study and demonstrate the efficiency and quality of our exact and heuristic solutions of OCS-G.

### 6.1 Experiment setup

We ran all the experiments on a PC with Intel Core2 at 2.67GHz, 4G memory running 32-bit Windows 7. All algorithms were implemented in C++. To directly show the performance of algorithms, we use $(k-1, 1)$ OCS and OCS-G models to conduct our experiments.

We use four real-world networks as our experiment datasets, and the statistics are shown in Table 2. Amazon is a product co-purchasing network of Amazon website. Nodes in Amazon represent products and if two products are frequently co-purchased, there exists an edge between them. DBLP is a scientific coauthor network extracted from a recent snapshot of the DBLP database.[1] Nodes in DBLP graph represent authors, and edges represent collaboration relationship. LiveJournal provides the LiveJournal friendship social network, it is a free online blogging community where users declare friendship. Friendster is an online gaming network. It used to be a social networking site where users can form friendship. We downloaded Amazon, LiveJournal, and Friendster from Stanford Large Network Dataset Collection.[2]

## 6.2 Performance

We first compare the performance of exact algorithms of basic OCS, optimized OCS, and Heuristic Strategy 1. For each $k$, we randomly select 100 nodes (with degree not less than $k-1$) for queries, and compare the average answering time. Because exact algorithms have exponential complexity, we terminate them when the running time exceeds 60s. The results of the three algorithms on the four networks are shown in Figure 4. We can see that our optimized OCS performs better than basic OCS, with about 3 to 27 times efficiency improvement, and the heuristic strategy overwhelms the two exact algorithms on performance by about two or three orders of magnitudes respectively. Actually, the superiority is more significant than Figure 4 shows. Because the maximum running time of exact algorithms is 60s in our setting. Especially for the biggest dataset Friendster, with millions of nodes, tens of millions of edges, and average degree 32.1, the executing time of heuristic strategy is around 100ms. This indicates that the heuristic strategy can support online search on large real networks. Note that as the average degree of dataset increases from Figure 4 a to d, the efficiency improvement of the optimized OCS decays, especially on the Friendster dataset. That is because as the average degree increase, a large portion of nodes in the dataset may be in multiple communities, thus the portion of boundary nodes increase and the portion of interior nodes decrease. So the filter mechanism of optimized OCS is limited, and that is the reason of the efficiency improvement decay.

Then, we compare the performance of exact algorithms of basic OCS-G, optimized OCS-G, and Heuristic Strategy 1. Note that basic OCS-G is directly iterating OCS for each node in a group query without any optimization, and optimized OCS-G refers to our exact solution for OCS-G. We set $k=5$ for Amazon, $k=7$ for DBLP, $k=9$ for LiveJournal, $k=11$ for Friendster, and change the group query size $|N|$. For each $|N|$, we test 20 randomly selected group queries, and compare the average time cost. Also, we terminate the exact algorithms after 600s. The results are shown in Figure 5. We can see that optimized OCS-G performs better than basic OCS-G, and as the query node number increases, the time cost of optimized OCS-G increases slowly than the basic algorithm. This indicates that our exact solution considering avoiding duplicate computations works well on OCS-G problem. Also, the Heuristic Strategy 1 won on the performance.

Figure 6 shows the performance of four heuristic strategies of OCS-G problem. We set the group query and $k$ value of different datasets as in Figure 5. We set the lower bound of

[1] http://dblp.uni-trier.de/xml/

[2] http://snap.stanford.edu/data/

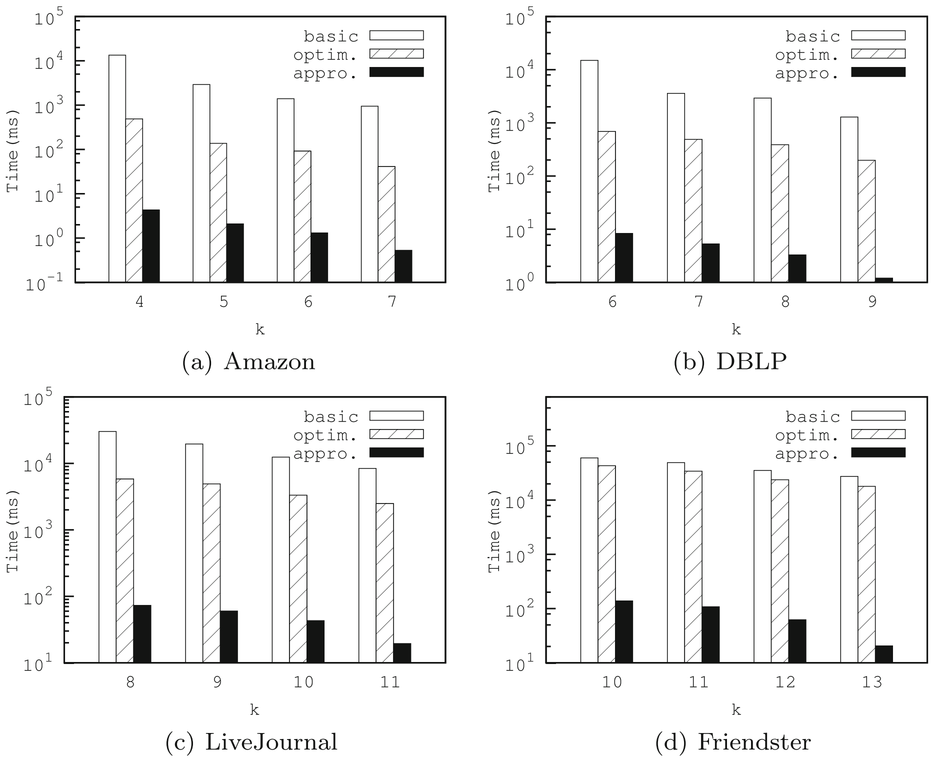


**Figure 4** Performance of basic OCS, optimized OCS, and Heuristic Strategy 1

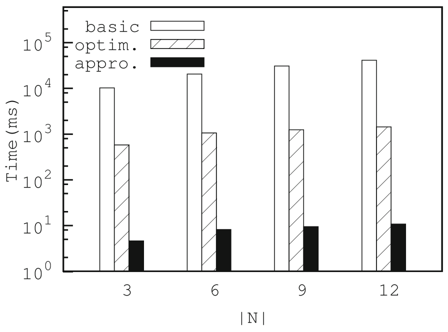
node degree restriction at $2(k-1)$ for Strategy 2 and Strategy 3, and set the tuning parameter $\tau = 2$ for Strategy 4. The results proof Proposition 1, the time cost of Strategy 1 to 3 is Strategy 3 < Strategy 1 < Strategy 2. And Strategy 4 performs the second best, with less than one time slower than Strategy 3.

## 6.3 Quality

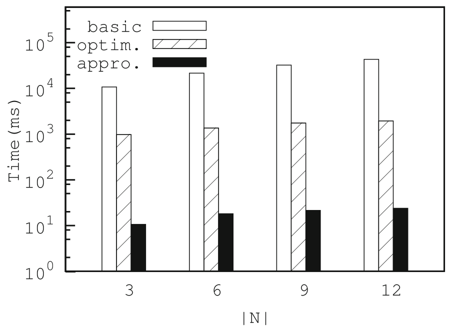
We compare the result quality of four heuristic strategies of OCS-G problem, and set the parameters and queries the same as in Figure 6. Clearly, each community in the approximate result is smaller than its corresponding community in the exact result. Let $R'$={$\mathbf{C'_1}, \ldots, \mathbf{C'_m}$} be the approximate result, and $\mathbf{C_i}$ be the exact community containing $\mathbf{C'_i}$, thus the accuracy of the approximate result $R'$ is defined as

$$Accuracy(R') = \frac{1}{m} \sum_{1 \le i \le m} \frac{\mathbf{C'_i}}{\mathbf{C_i}} \tag{6}$$

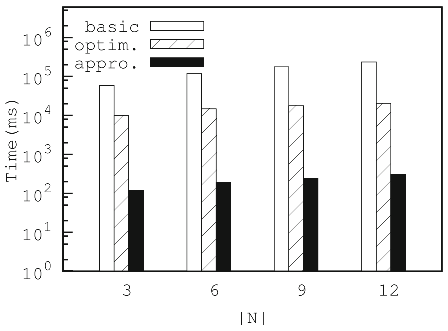
The average and variance accuracy of the four heuristic strategies are shown in Figure 7. It is clear that the accuracy of each strategy is over 50 %, and as our discussion in Proposition 1,



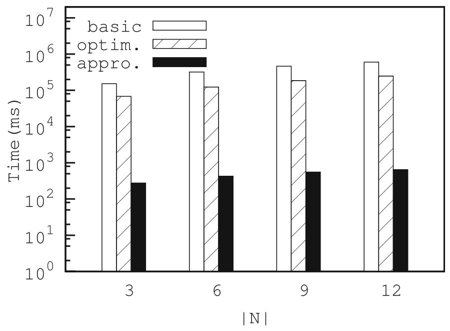
(a) Amazon(k=5)



(b) DBLP(k=7)



(c) LiveJournal(k=9)



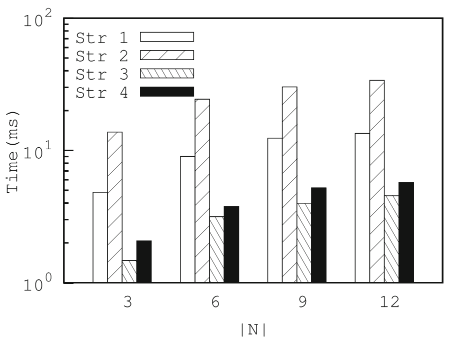
(d) Friendster(k=11)

**Figure 5** Performance of basic OCS-G, optimized OCS-G, and Heuristic Strategy 1

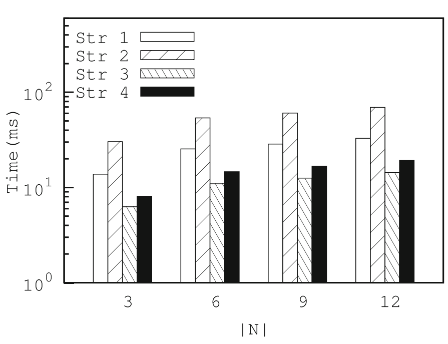
the accuracy of Strategy 1 to 3 is Strategy 2 > Strategy 1 > Strategy 3. Strategy 4 is the most accurate with about 90 % accuracy, and as the size of group query increases, the accuracy of Strategy 4 does not decay as much as the other three strategies, this proofs the effectiveness of discovery power parameter. Consider both efficiency and quality of the four heuristic strategies, Strategy 4 overwhelms the other three strategies.

## 6.4 Influence of node degree restriction

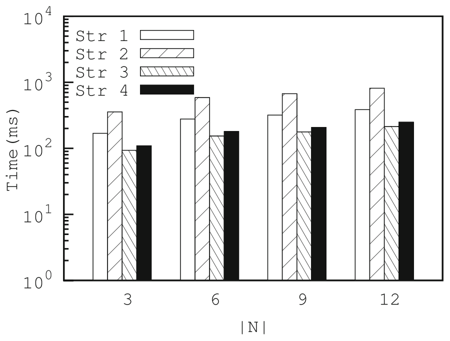
Now we investigate the influence of node degree restriction on Heuristic Strategy 2 and Strategy 3. For space limitation, we conduct experiments of OCS-G problem on DBLP dataset. We set $k = 7$, and the size of group query $|N| = 6$, we test 20 randomly selected group queries for different lower bounds of node degree, and compare the efficiency and quality of the strategies, the results are shown in Table 3. We can see that for both the two strategies, as the lower bound increases, the running time decreases sharply and the accuracy also decreases. However, the accuracy of Strategy 2 stays above 80 %, and the accuracy of Strategy 3 stays above 50 % with the efficiency highly improved. The results indicate that if the efficiency requirement is more important than the quality requirement, we could select Strategy 3, whose accuracy is acceptable.



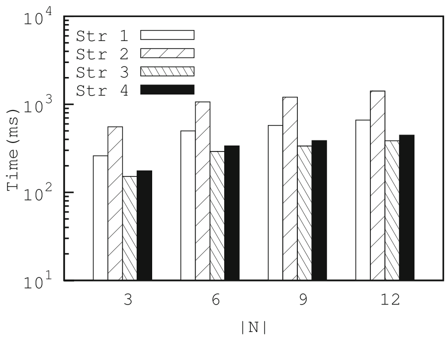
(a) Amazon(k=5)



(b) DBLP(k=7)



(c) LiveJournal(k=9)



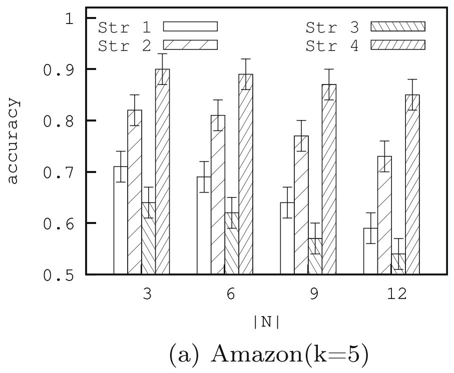
(d) Friendster(k=11)

**Figure 6** Performance of Heuristic Strategy 1, 2, 3, 4 of OCS-G

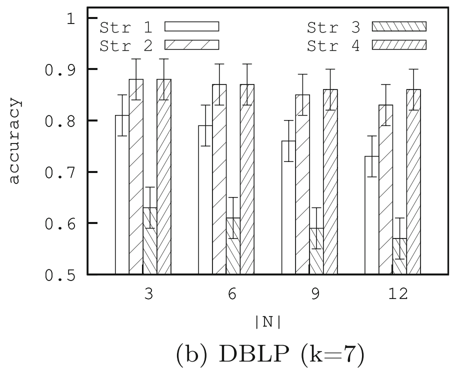
## 6.5 Influence of discovery power parameter

For the Heuristic Strategy 4, the discovery power parameter plays an important role on adjusting the efficiency and quality of the algorithm. We still conduct experiments of OCS-G problem on DBLP dataset. We set $k = 7$, and the size of group query $|N| = 6$, we test 20 randomly selected group queries for different $\tau$, and the results are shown in Table 4. The upper part shows the results using $\eta\tau$ as the threshold of discovery power, and the lower part shows the results using just $\tau$ as the threshold.

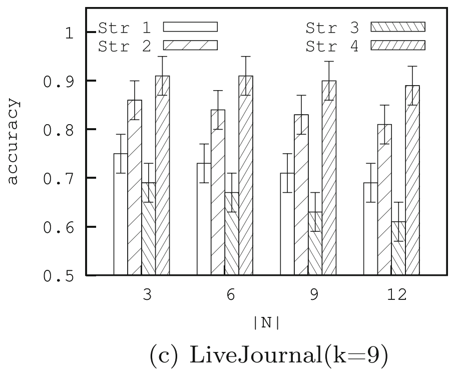
Comparing the two tables, we can see that the self-adapting parameter $\eta$ works well, as time costs are greatly improved with only small accuracy loss, and this proofs the effectiveness of our self-adapting mechanism. Besides, Table 4 shows that as $\tau$ increases, the time cost and accuracy decrease, and the adjusting effect of $\tau$ is remarkable. We could compare Table 4(upper part) with Table 3, to get the same accuracy 87 % for example, Strategy 4 takes 14.7ms, however Strategy 2 takes 46.5ms, which is over three times of Strategy 4; and at the same time cost about 14.7ms, Strategy 4 can get 87 % accuracy, while Strategy 3 can only get 61 % accuracy. This proofs that the tuning parameter $\tau$ of Strategy 4 is effective, it can better trade off the efficiency and quality of the heuristic strategy than node degree parameter. The result could also show that, to some extend, Strategy 4 is the most ideal heuristic strategy of the four strategies.



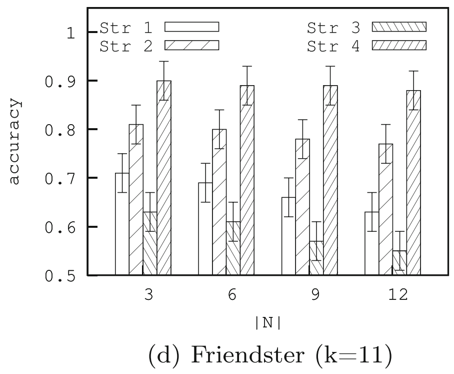
(a) Amazon(k=5)



(b) DBLP (k=7)



(c) LiveJournal(k=9)



(d) Friendster (k=11)

**Figure 7** Accuracy of Heuristic Strategy 1, 2, 3, 4 of OCS-G

**Table 2** Real-world networks for experiments

| Dataset | # Nodes | # Edges | Average degree |
|---|---|---|---|
| Amazon | 334,863 | 925,872 | 5.53 |
| DBLP | 968,956 | 4,826,365 | 9.96 |
| LiveJournal | 3,997,962 | 34,681,189 | 17.4 |
| Friendster | 4,856,981 | 78,051,642 | 32.1 |

**Table 3** Performance and quality of Strategy 2 and Strategy 3 for OCS-G on DBLP

| Lower bound | 8 | 11 | 14 | 17 |
|---|---|---|---|---|
| Time(ms) | 64.4 | 46.5 | 34.4 | 31.6 |
| Accuracy | 0.91 | 0.87 | 0.84 | 0.81 |
| | | | | |
| Time(ms) | 20.9 | 14.6 | 6.1 | 3.4 |
| Accuracy | 0.72 | 0.61 | 0.57 | 0.53 |

**Table 4** Performance and quality of Strategy 4 with and without self-adapting parameter $\eta$

| $\tau$ | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| `Time(ms)` | 28.9 | 14.7 | 6.3 | 5.2 | 4.3 |
| `Accuracy` | 0.93 | 0.87 | 0.79 | 0.73 | 0.63 |
| | | | | | |
| `Time(ms)` | 35.3 | 22.5 | 15.3 | 8.7 | 6.5 |
| `Accuracy` | 0.94 | 0.91 | 0.85 | 0.80 | 0.75 |

## 7 Conclusion

In this paper we studied a framework of overlapping community search for group query, including both exact and heuristic solutions. The performance of exact solution was highly improved by boundary node limitation and avoiding repeated computations. Besides, we proposed four heuristic strategies, some of which are adjustable by utilizing node degree and discovery power parameters, thus they could satisfy different efficiency and quality requirements. Comprehensive experiments were conducted to evaluate the efficiency of the optimized exact algorithms, and the efficiency and quality differences of the four heuristic strategies, and the influence of the adjusting parameters. Through the experiments we demonstrated that our solutions were effective and efficient to discover overlapping communities for group query in real networks, and the heuristic strategies are flexible for different requirements.

**Acknowledgments** This work is supported by the National Basic Research 973 Program of China under Grant No.2012CB316201, the National Natural Science Foundation of China under Grant Nos. 61033007, 61472070, and the Fundamental Research Funds for the Central Universities (N120816001).

## References

1. Adamcsek, B., Palla, G., Farkas, I.J., Derényi, I., Vicsek, T.: Cfinder: locating cliques and overlapping modules in biological networks. Bioinformatics **22**(8), 1021–1023 (2006)
2. Ahn, Y.Y., Bagrow, J.P., Lehmann, S.: Link communities reveal multiscale complexity in networks. Nature **466**(7307), 761–764 (2010)
3. Bagrow, J.P.: Evaluating local community methods in networks. J. Stat. Mech. Theory Exp. **2008**(05), P05001 (2008)
4. Bagrow, J.P., Bollt, E.M.: Local method for detecting communities. Phys. Rev. E **72**(4), 046108 (2005)
5. Brunato, M., Hoos, H., Battiti, R.: On effectively finding maximal quasi-cliques in graphs. In: Learning and Intelligent Optimization, pp. 41–55. Springer, Berlin (2008)
6. Chen, J., Zaïane, O., Goebel, R.: Local community identification in social networks. In: International Conference on Advances in Social Network Analysis and Mining, 2009. ASONAM'09, pp. 237–242. IEEE (2009)
7. Clauset, A.: Finding local community structure in networks. Phys. Rev. E **72**(2), 026132 (2005)
8. Cui, W., Xiao, Y., Wang, H., Lu, Y., Wang, W.: Online search of overlapping communities. In: Proceedings of the 2013 International Conference on Management of Data, pp. 277–288. ACM (2013)
9. Dourisboure, Y., Geraci, F., Pellegrini, M.: Extraction and classification of dense communities in the web. In: Proceedings of the 16th International Conference on World Wide Web, pp. 461–470. ACM (2007)

10. Evans, T., Lambiotte, R.: Line graphs, link partitions, and overlapping communities. Phys. Rev. E **80**(1), 016105 (2009)
11. Flake, G.W., Lawrence, S., Giles, C.L.: Efficient identification of web communities. In: Proceedings of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 150–160. ACM (2000)
12. Gibson, D., Kumar, R., Tomkins, A.: Discovering large dense subgraphs in massive graphs. In: Proceedings of the 31st International Conference on Very Large Data Bases, pp. 721–732. VLDB Endowment (2005)
13. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proc. Natl. Acad. Sci. **99**(12), 7821–7826 (2002)
14. Gregory, S.: Finding overlapping communities in networks by label propagation. New J. Phys. **12**(10), 103018 (2010)
15. Herrera, M., Roberts, D.C., Gulbahce, N.: Mapping the evolution of scientific fields. PloS one **5**(5), e10355 (2010)
16. Jonsson, P.F., Cavanna, T., Zicha, D., Bates, P.A.: Cluster analysis of networks generated through homology: automatic identification of important protein communities involved in cancer metastasis. BMC Bioinformatics **7**(1), 2 (2006)
17. Leskovec, J., Lang, K.J., Dasgupta, A., Mahoney, M.W.: Statistical properties of community structure in large social and information networks. In: Proceedings of the 17th International Conference on World Wide Web, pp. 695–704. ACM (2008)
18. Lim, S., Ryu, S., Kwon, S., Jung, K., Lee, J.G.: Linkscan*: Overlapping community detection using the link-space transformation. In: 2014 IEEE 30th International Conference on Data Engineering (ICDE), pp. 292–303. IEEE (2014)
19. Luo, F., Wang, J.Z., Promislow, E.: Exploring local community structures in large networks. Web Intelligence and Agent Systems **6**(4), 387–400 (2008)
20. Palla, G., Derényi, I., Farkas, I., Vicsek, T.: Uncovering the overlapping community structure of complex networks in nature and society. Nature **435**(7043), 814–818 (2005)
21. Papadopoulos, S., Kompatsiaris, Y., Vakali, A., Spyridonos, P.: Community detection in social media. Data Min. Knowl. Disc. **24**(3), 515–554 (2012)
22. Papadopoulos, S., Skusa, A., Vakali, A., Kompatsiaris, Y., Wagner, N.: Bridge bounding: a local approach for efficient community discovery in complex networks (2009). arXiv:0902.0871
23. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Phys. Rev. E **76**(3), 036106 (2007)
24. Shan, J., Shen, D., Nie, T., Kou, Y., Yu, G.: An efficient approach of overlapping communities search. In: Database Systems for Advanced Applications, pp. 374–388. Springer (2015)
25. Sozio, M., Gionis, A.: 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, pp. 939–948. ACM (2010)
26. Šubelj, L., Bajec, M.: Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction. Phys. Rev. E **83**(3), 036103 (2011)