An Efficient Approach of Overlapping Communities Search

Jing Shan(✉), Derong Shen, Tiezheng Nie, Yue Kou, and Ge Yu

College of Information Science and Engineering, Northeastern University, Liaoning 110004, China
mavisshan0129@gmail.com,
{shenderong,nietiezheng,kouyue,yuge}@ise.neu.edu.cn

Abstract. A great deal of research has been dedicated to discover overlapping communities, as in most real life networks such as social networks and biology networks, a node often involves in multiple overlapping communities. However, most work has focused on community detection, which takes the whole graph 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 given nodes is a flexible and light-weight solution, and also supports dynamic graphs very well. Thus, in this paper, we study an efficient solution for overlapping community search problem. We propose an exact algorithm whose performance is highly improved by considering boundary node limitation and avoiding duplicate computations of multiple input nodes, and we also propose three approximate strategies which trade off the efficiency and quality, and can be adopted in different requirements. Comprehensive experiments are conducted and demonstrate the efficiency and quality of the proposed algorithms.

1 Introduction

Community structure [9] is observed commonly existing in networks such as social media and biology. 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, finding communities 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) [13,10,8,11]

M. Renz et al. (Eds.): DASFAA 2015, Part I, LNCS 9049, pp. 374–388, 2015.
DOI: 10.1007/978-3-319-18120-2_22

has drawn a lot of attention in recent years. OCD dedicates to find all overlapping communities of the entire network, which has some shortcomings in some applications: First, it is time consuming when a network is quite large. Second, OCD uses a global criterion to find communities for all nodes in a network, which is unappropriate 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. [7].

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. In [7], an algorithm of finding overlapping communities of one query node was proposed, but it still has a large room for performance improvement. Besides, in some scenarios, the OCS query includes a set of nodes. For example, suppose a piece of news published on social network was read by a group of people, the service provider wants push the news to user communities in which people will be also interested in this news; or a product has been bought by a group of customers, and the producer wants to investigate the consumer groups in which people 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 efficient approach for overlapping community search which not only highly improves the performance of single-node overlapping community search, but also includes an efficient framework for multiple-node query. In summary, we make the following contributions:

- We introduce the definition of boundary node, and use boundary node limitation to highly improve the performance of single-node overlapping community search algorithm.
- We propose a framework for multiple-node overlapping community search and try our best to avoid waste computations by strongly theoretical supports.
- We also propose a series of approximate strategies which trade off the efficiency and quality to suit different requirements.
- 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. In Section 4 we introduce both exact and approximate algorithms. We present our experimental results in Section 5. Finally, Section 6 concludes the paper.

2 Related Work

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

Palla et al. first addressed overlapping community structure existing in most real networks [13], 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. Besides structure based method like CPM, overlapping community detection could also be modeled as link-partition problem [8,2,11]. 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 [8], Jaccard-type similarity computation [2], or density-based clustering algorithm SCAN [11], 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 OCD [10] [16], 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 query nodes, but not the whole graph, thus it is more appropriate for online query.

Considering the scalability problem of community detection, local community detection problem, also called community search, has also received a lot of attention [6,12,14,5]. These methods start from a seed node or node set, and then attach adjacent nodes to community as long as these nodes can increase some community quality metrics such as local modularity [6], subgraph modularity [12], or node outwardness [3]. In [15], 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 discover overlapping communities, they can just find one community.

Our work is inspired by Cui et al [7], 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 solution of overlapping community search for multiple nodes. Although simply iterating the algorithm in [7] could solve the problem, this method could produce a lot of waste computations and it is not an effective solution. Thus, we propose an efficient approach for OCS, considering both single query node and multiple query nodes situations.

3 Problem Definition

In this section, we define the problem of overlapping community search more formally, including OCS for both single query node and multiple query nodes.

Intuitively, a typical member of a community is linked with many but not necessarily all other nodes in the community, so we use k-clique as building

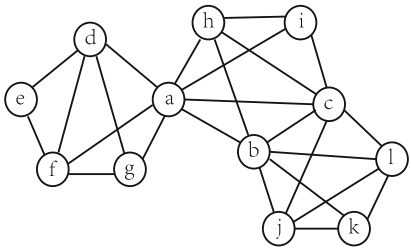


Fig. 1. A toy social network graph

blocks of a community to reflect this characteristic, just as Palla et al. [13]. 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. [7] proposed a less strict definition: two k-cliques are adjacent if they share at least α nodes, where $\alpha \leq k-1$; and k-clique can be replaced by γ-*quasi-k-clique* [4] 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 multiple query nodes.

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

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

Example 1 ((α,γ)-OCS-M). For the graph in Fig. 1, given a set of query nodes $V_q = \{\text{a,b,c}\}$, let $k=4$, consider $(3,1)$-OCS-M, we get three communities $\{a,d,g,f\}$, $\{a,b,c,h,i\}$, $\{b,c,j,k,l\}$.

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

4 Overlapping Community Search Algorithms

We first propose a naive algorithm derived from OCS to solve OCS-M, then we propose optimized OCS and OCS-M algorithms based on a series of theorems with strong proofs. At last, we propose a series of approximate strategies to make the process more efficient.

4.1 Naive Algorithm

OCS algorithm searches overlapping communities of one single input node, intuitively, when given a set of nodes as input, we could iterate the OCS algorithm for each node, hence we get the naive algorithm of OCS-M as depicted in Alg. 1. 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 it starts from. Thus, to avoid redundant enumeration, we only enumerate unvisited cliques for each round of iteration.

Algorithm 1. Naive OCS-M

Input: $G(V, E)$, V_q, α, γ, k;
Output: The overlapping communities containing $\forall v_i \in V_q$

```
1 R ← ∅;
2 foreach v_i ∈ V_q do
3     while C ← next_clique(v_i), C ≠ ∅ do
4         C ← expand(C);
          // find the clique component C of C
5         R ← R ∪ C;
6 Return R;
```

We adopt the same depth-first-search strategy for *next_clique*() and *expand*() as in [7]. We omit the details and refer readers who are interested to [7]. Consider Example 1, when the input node is a, *next_clique*(a) may first return *Clique* $adfg$, and then *expand*() on this clique gets *Community* $\{a, d, f, g\}$, the next call of *next_clique*(a) may return $abch$, and *expand*() gets $acih$, thus we get *Community* $\{a, b, c, i, h\}$. Further call of *next_clique*(a) will not return any more cliques because all cliques containing a have been visited. When input node comes to b, we get $bclj$, and calling *expand*() brings us $blkj$, thus we get *Community* $\{b, c, l, k, j\}$. Consider input node c, no further result is derived because all cliques containing c have been visited, and the procedure terminates.

4.2 Optimized OCS Algorithm

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 [7], 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. Considering the three types of node definitions, we propose a theorem which could be used to optimize the OCS algorithm.

Theorem 1. *For OCS algorithm, one community can be derived by only expanding boundary nodes or 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. □

Algorithm 2. optimized next_clique(v_0)

Input: v_0: a query node
Output: C: next γ-quasi-k-clique

```
1  U ← {v0};
2  DFS (U, v0);
3  Procedure DFS(U, u)
4      if |U| = k then
5          if U is a γ-quasi-k-clique and U is unvisited then
6              return U;
7          else
8              return;
9      if g(U) < γ k(k−1)/2 then
10         return;
11     foreach (u, v) ∈ E, v ∉ U do
12         if neighbor(v) ≥ ⌈γ C(k,2) − C(k−1,2)⌉ then
13             DFS(U ∪ {v}, v)
```

According to Thm. 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 γ-quasi-k-clique, the minimum degree of a node should be $\lceil \gamma \binom{k}{2} - \binom{k-1}{2} \rceil$. Base on the definition of community, if one node has

less than $\lceil \gamma\binom{k}{2} - \binom{k-1}{2} \rceil$ 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 Alg. 2 and Alg. 3.

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 (Alg.2 line 4-8), we use node degree condition to prune nodes (line 12), $neighbor(v)$ represents the degree of v. Besides, $g(U)$ is another pruning condition proposed in the original OCS Algorithm [7] (line 9), it represents the maximal number of edges that the resulting 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 γ-quasi-k-clique
   Output: A: the community of C
1  A ← C;
2  Expand_Clique (C);
3  return A;
4  Procedure Expand_Clique(C)
5      sort C by d_nc(n), n ∈ C;
6      foreach S_1 ∈ C and |S_1| ≥ α do
7          S_2 = C − S_1;
8          foreach u ∈ S_1 do
9              foreach v ∈ neighbor(u) do
10                 if d_nc(v) > 0 and neighbor(v) ≥ ⌈γ(k choose 2) − (k−1 choose 2)⌉ then
11                     Cand ← Cand ∪ v;
12         if |Cand| ≤ |S_2| or g(S_1) < γ k(k−1)/2 then
13             Continue;
14         foreach S'_2 ∈ Cand, |S_2| = |S'_2| do
15             C' ← S_1 ∪ S'_2;
16             if C' is unvisited and C' is a γ-quasi-k-clique then
17                 A ← A ∪ 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) degree should be not less than the lower bound (line 10). Note that $d_{nc}(v)$ denotes the number of v's neighbors which are not in the community, $d_{nc}(v) = 0$ means v is an interior node of the current explored community. Notice 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 node degree at the beginning. For a 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} value 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 Sec. 5.

4.3 Optimized OCS-M Algorithm

When it comes to OCS-M 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 (Consistency). In (α, γ)-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.

Consider Example 1, 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 $\{a, b, c, i, h\}$. 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 which nodes in the existing community can be skipped and which can not, and we get Thm. 2.

Theorem 2. *For $(k-1, 1)$-OCS-M, given a node v which is a member of existing community $\mathbf{C_m}$, and 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: $vn'_1 \dots n'_{k-1}$ which belongs to $\mathbf{C'_m}$, and the degree of v in C'_l is $d_{C'_l}(v) = |n'_1 \dots n'_{k-1}| = k - 1$, and we know that $v \in \mathbf{C_m}$, so there exists a clique C_l: $vn_1 \dots n_{k-1}$ which belongs to $\mathbf{C_m}$ and $d_{C_l}(v) = |n_1 \dots n_{k-1}| = k - 1$. 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 Thm. 2, we could easily infer that for $(k-1,1)$-OCS-M, if a node already exists in a community and its degree is not larger than k, it can be skipped. 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 (α,γ)-OCS-M. For a γ-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 α nodes. Thus, we replace the conditions in the proof of Thm. 2 and get Thm. 3.

Theorem 3. *For (α,γ)-OCS-M, given a node v which is a member of existing community $\mathbf{C_m}$, and 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 2 (Optimized-(α,γ)-OCS-M). For the graph in Fig. 1, suppose input node set V_q={a,b,c}, let $k = 5$, consider $(3, 0.9)$-OCS-M, 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 Thm. 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 2 we can see that utilizing Thm. 3, the performance of OCS-M Algorithm is remarkably improved. However, taking $(k-1,1)$-OCS-M 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. Base on the definitions of interior and boundary node, we have Thm. 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. □

According to Thm. 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. Utilizing Thm. 3 and Thm. 4, we could modify Alg. 2 by replacing line 12-13 with Alg. 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-M, if the node already

exists in a community, we use the lower bound mentioned in Thm. 3 as pruning condition (line 3); if it does not exist in a community, we use the lower bound of node degree mentioned in Alg. 2 (line 6).

Algorithm 4. modify next_clique(v_0)

```
1 if d_nc(v) > 0 then
2     if v ∈ R then
3         if neighbor(v) > max{2⌈γ(k choose 2) − (k−1 choose 2)⌉ − (α − 1), ⌈γ(k choose 2) − (k−1 choose 2)⌉} then
4             DFS(U ∪ v, v)
5     else
6         if neighbor(v) ≥ ⌈γ(k choose 2) − (k−1 choose 2)⌉ then
7             DFS(U ∪ v, v)
```

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

After modifying Alg. 2 and Alg. 3, we get the optimized algorithm of OCS-M, the improvement of performance will be shown through experiments in Sec. 5.

4.4 Approximate Strategies

Although the performance of the exact algorithm has been greatly improved, it is still an NP-hard problem. Thus, we propose a series of approximate strategies which could trade off the performance and quality of our OCS and OCS-M algorithms. We use two conditions boundary node and node degree to adjust the efficiency and quality of the algorithm.

Considering the search process of one community $\mathbf{C_m}$, it starts from a query node, and adjacent nodes are added into $\mathbf{C_m}$ as long as they satisfy that 1) they belong to a γ-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 scatter in it, if we traverse nodes out of the circle, but not wander in the circle, the entire community can be found more earlier. According to Thm. 1 and Thm. 4, interior nodes (i.e. nodes in the circle) can be omitted without losing the completeness, if we traverse the boundary nodes selectively or only traverse the exterior nodes, the search process could be terminated earlier with sacrificing result quality.

Besides, we also consider node degree as another traversing condition. 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 $neighbor(v)$ in both $next\_clique()$ and $expand()$, the higher the lower bound is, the less the traversed nodes are, and the more the quality loss is.

With boundary node and node degree as traversing conditions, we could form three approximate strategies:

- *Strategy 1*: traverse boundary nodes with node degree restriction, and all exterior nodes. That means we partially traverse boundary nodes, and completely traverse exterior nodes.
- *Strategy 2*: traverse only exterior nodes. That means we skip all boundary nodes. By doing this, we could guarantee that if one node belongs to the community, it is only traversed once, no repetitive traversal is made.
- *Strategy 3*: traverse only exterior nodes with node degree restriction. That means we skip all boundary nodes, and partially traverse exterior nodes.

Note that we only apply our approximate strategies on the *expand*() procedure, the *next_clique*() procedure is still exact. Because compared to expanding seed cliques, the computations of finding a new clique as a 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 approximate results, we only apply it on the *expand*() procedure.

Consider a k-clique community $\mathbf{C_m}$, the exact algorithm will explore $O(\binom{|\mathbf{C_m}|}{k})$ cliques. For Strategy 2, each time when a new node is found, a clique will be visited. Thus, it will only explore $O(|\mathbf{C_m}|)$ cliques. For Strategy 1, 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. Thus Strategy 1 will explore $O(\binom{n}{k} + |\mathbf{C_m}|)$ cliques. For Strategy 3, suppose the number of exterior nodes which exceed the lower bound of node degree is n, then it will explore $O(n)$ cliques. In this way, for Strategy 2 and 3, we reduce the exponential complexity to linear, and the efficiency is highly improved; for Strategy 1, the result is the most accurate of the three. Theoretically, the relationship of efficiency and quality of these three strategies is depicted in Lemma 1, we will demonstrate it by experiments in Sec. 5.

Lemma 1. *For the three approximate strategies of OCS and OCS-M, the efficiency of them is Strategy* $1 <$ *Strategy* $2 <$ *Strategy* 3*, and the quality of them is Strategy* $1 >$ *Strategry* $2 >$ *Strategy* 3*.*

5 Experimental Study

In this section, we present experimental study and demonstrate the efficiency and quality of our OCS and OCS-M algorithms.

5.1 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 intuitively show the performance of algorithms, we use $(k-1, 1)$ OCS and OCS-M models to conduct our experiments.

We use three real-world networks as our experiment datasets, and the statistics are shown in Table 1. Amazon is a product co-purchasing network of Amazon

Table 1. 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

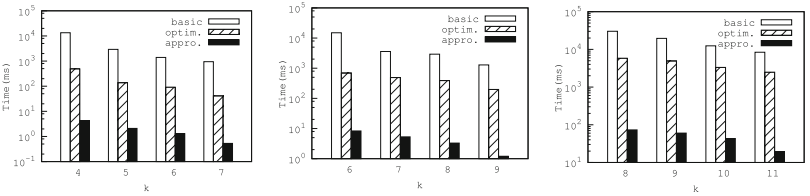
website[1]. Nodes in Amazon represent products and if two products are frequently co-purchased, there exists an edge between them. DBLP is a scientific coauthor network extracted from a recent snapshot of the DBLP database[2]. Nodes in DBLP graph represent authors, and edges represent collaboration relationship. LiveJournal provides the LiveJournal friendship social network[3], it is a free online blogging community where users declare friendship.

5.2 Performance

We first compare the performance of exact algorithms of basic OCS, optimized OCS, and approximate Strategy 2. 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 exeeds 60s. The results of the three algorithms on the three networks are shown in Fig. 2. We can see that our optimized OCS performs better than basic OCS, with about 20 times efficiency improvement, and the approximate strategy overwhelms the two exact algorithms on performance by about two or three orders of magnitudes respectively. Actually, the superiority is more significant than Fig. 2 shows. Because the maximal running time of exact algorithms is 60s in our setting. Especially for the biggest dataset LiveJournal, with millions of nodes, tens of millions of edges, and average degree 17.3, the executing time of approximate strategy is less than 100ms. This indicates that the approximate strategy can support online search on large real networks.

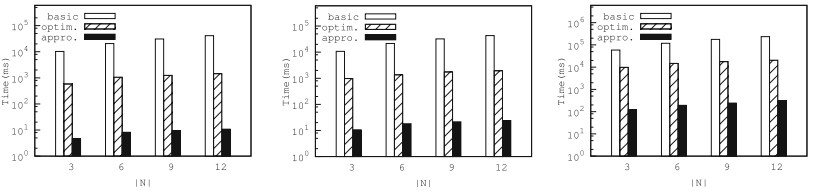
Then, we compare the performance of exact algorithms of basic OCS-M, optimized OCS-M, and approximate Strategy 2. We set $k = 5$ for Amazon, $k = 7$ for DBLP, $k = 9$ for LiveJournal, and change the query node number $|N|$ of query sets. For each $|N|$, we test 20 randomly selected query sets, and compare the average time cost. Also, we terminate the exact algorithms after 600s. The results are shown in Fig. 3. We can see that optimized OCS-M performs better than basic OCS-M, and as the query node number increases, the time cost of optimized OCS-M increases slowly than the basic algorithm. This indicates that our optimized algorithm avoiding duplicate computations works well on OCS-M problem. Also, the approximate Strategy 2 won on performance.

[1] http://snap.stanford.edu/data/com-Amazon.html
[2] http://dblp.uni-trier.de/xml/
[3] http://snap.stanford.edu/data/com-LiveJournal.html



(a) Amazon (b) DBLP (c) LiveJournal

Fig. 2. Performance of basic OCS, optimized OCS, and approximate Strategy 2



(a) Amazon(k=5) (b) DBLP(k=7) (c) LiveJournal(k=9)

Fig. 3. Performance of basic OCS-M, optimized OCS-M, and approximate Strategy 2

5.3 Quality

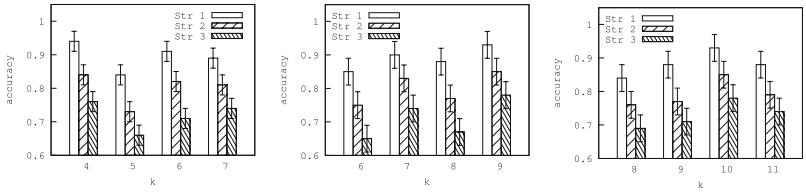
We compare the result quality of three approximate strategies of OCS problem, for OCS-M problem the situation is similar, thus we save the comparison for space limitation. We set the lower bound of node degree restriction at $2(k-1)$ for Strategy 1 and Strategy 3, and randomly select 100 valid query nodes for different k. 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

$$Accuracy(R') = \frac{1}{m} \sum_{1 \le i \le m} \frac{\mathbf{C'_i}}{\mathbf{C_i}} \tag{2}$$

The average and variance accuracy of the three approximate strategies are shown in Fig. 4. It is clear that all the three strategies' accuracy is over 60%, and as our discussion in Lemma 1, Strategy 1 with more than 80% accuracy performs best on quality, and Strategy 2 is better than Strategy 3.

5.4 Influence of Node Degree Restriction

Now we investigate the influence of node degree restriction on approximate Strategy 1 and Strategy 3. For space limitation, we conduct experiments of OCS problem on DBLP dataset. We set $k = 7$, randomly select 100 valid query nodes for



(a) Amazon (b) DBLP (c) LiveJournal

Fig. 4. Accuracy of approximate Strategy 1, 2, 3

different lower bounds of node degree , and compare the efficiency and quality of the algorithms, the results are shown in Table 2. We can see that for both of the two strategies, as the lower bound increases, the running time decreases sharply and the accuracy also decreases. However, the accuracy of Strategy 1 stays above 85%, and the accuracy of Strategy 3 stays above 50% with the efficiency improved 6 times. The results indicate that if the quality requirement is more important than the efficiency requirement, we could select Strategy 1, for the opposite situation, we could select Strategy 3, whose accuracy is also acceptable.

Table 2. Performance and Quality of Strategy 1 and Strategy 3

Lower Bound	8	11	14	17
Time(ms)	14.8	10.2	7.5	6.9
Accuracy	0.95	0.92	0.88	0.85

Lower Bound	8	11	14	17
Time(ms)	4.8	3.1	13	0.8
Accuracy	0.80	0.77	0.69	0.58

6 Conclusion

In this paper we studied an efficient solution for overlapping community search problem. We proposed an exact algorithm whose performance was highly improved for both single node overlapping community search and multiple nodes overlapping community search with strong theoretical supports. Besides, we proposed three approximate strategies which 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 difference of the three approximate strategies. Through the experiments we demonstrated that our solutions were effective and efficient to discover overlapping communities in real networks, and the approximate strategies are flexible for different requirements.

Acknowledgments. This work is supported by the National Basic Research 973 Program of China under Grant (2012CB316201) and the National Natural Science Foundation of China under Grant (61472070).

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. Journal of Statistical Mechanics: Theory and Experiment **2008**(05), P05001 (2008)
4. Brunato, M., Hoos, H.H., Battiti, R.: On effectively finding maximal quasi-cliques in graphs. In: Maniezzo, V., Battiti, R., Watson, J.-P. (eds.) LION 2007 II. LNCS, vol. 5313, pp. 41–55. Springer, Heidelberg (2008)
5. Chen, J., Zaïane, O., Goebel, R.: Local community identification in social networks. In: International Conference on Advances in Social Network Analysis and Mining, ASONAM 2009, pp. 237–242. IEEE (2009)
6. Clauset, A.: Finding local community structure in networks. Physical Review E **72**(2), 026132 (2005)
7. 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)
8. Evans, T., Lambiotte, R.: Line graphs, link partitions, and overlapping communities. Physical Review E **80**(1), 016105 (2009)
9. Girvan, M., Newman, M.E.: Community structure in social and biological networks. Proceedings of the National Academy of Sciences **99**(12), 7821–7826 (2002)
10. Gregory, S.: Finding overlapping communities in networks by label propagation. New Journal of Physics **12**(10), 103018 (2010)
11. 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)
12. Luo, F., Wang, J.Z., Promislow, E.: Exploring local community structures in large networks. Web Intelligence and Agent Systems **6**(4), 387–400 (2008)
13. 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)
14. Papadopoulos, S., Skusa, A., Vakali, A., Kompatsiaris, Y., Wagner, N.: Bridge bounding: A local approach for efficient community discovery in complex networks. arXiv preprint arXiv:0902.0871 (2009)
15. 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)
16. Šubelj, L., Bajec, M.: Unfolding communities in large complex networks: Combining defensive and offensive label propagation for core extraction. Physical Review E **83**(3), 036103 (2011)