Online Search of Overlapping Communities

Wanyun Cui
Fudan University
wanyuncui1@gmail.com

Yanghua Xiao*
Fudan University
shawyh@fudan.edu.cn

Haixun Wang
Microsoft Research Asia
haixunw@microsoft.com

Yiqi Lu
Fudan University
luyiqi@gmail.com

Wei Wang
Fudan University
weiwang1@fudan.edu.cn

ABSTRACT

A great deal of research has been conducted on modeling and discovering communities in complex networks. In most real life networks, an object often participates in multiple overlapping communities. In view of this, recent research has focused on mining overlapping communities in complex networks. The algorithms essentially materialize a snapshot of the overlapping communities in the network. This approach has three drawbacks, however. First, the mining algorithm uses the same global criterion to decide whether a subgraph qualifies as a community. In other words, the criterion is fixed and predetermined. But in reality, communities for different vertices may have very different characteristics. Second, it is costly, time consuming, and often unnecessary to find communities for an entire network. Third, the approach does not support dynamically evolving networks. In this paper, we focus on online search of overlapping communities, that is, given a query vertex, we find meaningful overlapping communities the vertex belongs to in an online manner. In doing so, each search can use community criterion tailored for the vertex in the search. To support this approach, we introduce a novel model for overlapping communities, and we provide theoretical guidelines for tuning the model. We present several algorithms for online overlapping community search and we conduct comprehensive experiments to demonstrate the effectiveness of the model and the algorithms. We also suggest many potential applications of our model and algorithms.

Categories and Subject Descriptors

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

General Terms

Algorithms, Theory, Experimentation

Keywords

Community search; Social networks; Graph mining

*Correspondence author. This work was supported by the National NSFC (No. 61003001, 61170006, 61171132, 61033010); Specialized Research Fund for the Doctoral Program of Higher Education No. 20100071120032; NSF of Jiangsu Province (No. BK2010280).

SIGMOD'13, June 22–27, 2013, New York, New York, USA.

1. INTRODUCTION

Most complex networks in nature and human society contain *community structures* that serve as functional building blocks for the networks. Furthermore, communities often overlap with each other, that is, a vertex in a network may belong to more than one community.

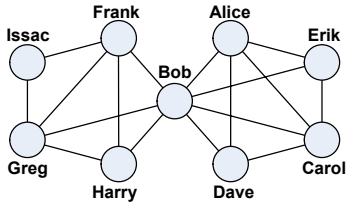


Figure 1: A toy social graph

1.1 Overlapping Communities

Overlapping community structures can be observed in many real networks, including social networks, biology networks, and sematic networks, and they are vital in revealing the internal structure of large networks [1]. The ubiquity of overlapping communities signifies the importance of finding overlapping communities in graphs.

Communities can be defined in many ways. In general, a community refers to a group of vertices that are densely connected to each other and sparsely connected to other vertices in the graph. The density of a community usually is measured by its average degree. In this paper, we use k-cliques [1] as the building blocks of a community. We will show the rational in Section 3. A k-clique is a complete graph of k vertices. Given a graph G, we can find all k-cliques in G. We say two k-cliques are adjacent if they share $k-1$ vertices in G. We can then create a k-clique graph for G. In the k-clique graph, each vertex is a k-clique and an edge between two vertices means the two k-cliques they represent are adjacent in G. A community is defined as a connected component, called a k-clique component, in the k-clique graph. Clearly, the larger the k value, the denser the communities. By tuning k, we obtain communities of different density. Next, we use a toy social network to illustrate overlapping communities.

EXAMPLE 1 (OVERLAPPING COMMUNITIES). *Consider the toy social graph shown in Figure 1. Suppose we need to find the communities containing Bob. When $k = 3$, we get two overlapping communities: $abcde$[1] and $bfghi$ (for simplicity, we use the first character of a person's name to represent the person). When $k = 4$, we find two communities: $abcde$ and $bfhg$. $bfhg$ is denser than*

[1]In the following text, we use 'abcde' to denote the clique consisting of vertex set $\{a, b, c, d, e\}$

bfhgi because Issac, who has fewer connections to the community, is excluded.

1.2 OCD vs OCS

There are two related problems: *overlapping community detection* (OCD), which finds overlapping communities in the entire network, and *overlapping community search* (OCS), which finds overlapping communities that a specific vertex belongs to.

The difference between OCD and OCS is clear. For a given graph, OCD takes k as input, and finds overlapping communities in the entire graph, using a batch process. As a result, all detected communities are of the density that is not less than the same threshold. On the other hand, OCS takes a vertex and a specific k as input, and finds overlapping communities containing the vertex, using an online process.

One common operation in community analysis is to retrieve the communities containing a vertex. To support online query answering by OCD, we need to detect and materialize all communities in an offline pre-computation stage and then index the communities. In contrast, with OCS, we just need to specify the vertex of interest in the query. Using OCD to support this common operation is less desirable for large graphs due to the following reasons.

- First, *it is costly and time consuming to find communities for the entire graph by OCD*. OCD in general is NP-hard, and many real life graphs are quite large, with millions or even billions of nodes. For example, the friendship network of Facebook [2] contains over 800 million nodes and 100 billion links. As a result, OCD is computationally prohibitive on real large graphs. In contrast, with OCS, we just need to find communities within the local neighborhoods of the vertex. In the experimental section, we will show that in most cases, our OCS solution only needs several milliseconds to find answers, independent of the entire graph size.

- Second, *OCD uses a global criterion for community detection for all vertices in a network*. The semantics of the discovered community in general heavily depends on the parameters. Example 1 has shown that for the same vertex, a different k leads to communities of different density and different semantics. Results on real networks (the details are presented in the experiment section) further show that under a given k, some vertices may have no corresponding valid communities. In other words, a fixed k is not appropriate for all vertices. OCD does not support using the most appropriate k for each individual vertex.

- Third, *it is difficult for OCD to support dynamically evolving graphs*. Graphs in real life are always evolving over time. OCD can only be used for offline analysis. Generally, we cannot afford to run OCD very frequently. As a result, a solution of OCD usually loses its freshness and effectiveness after a short period of time. In contrast, with OCS, we are free to issue queries for any vertex, which will not significantly hinder system performance.

In summary, OCD plus indexing suffers when the graph is dynamically evolving or users want to tune the density parameters for a better result. Because it is costly to rerun OCD and rebuild the index. Unfortunately, most real networks, such as Facebook friendship networks, are evolving and users in these networks usually expect to freely explore his or her neighborhood community. This motivates us to study online OCS.

1.3 Challenges

To understand the major challenges of OCS, we compare OCS with a related and well studied problem: *Community Search* (CS). Given a vertex (or a set of vertices), community search finds the community (instead of a set of overlapping communities) that contains the vertex [3]. In the following analysis, we show that CS and OCS are fundamentally different, and methods of CS cannot be directly applied to solve the OCS problem. The challenges of OCS are summarized as the follows:

- *Our first challenge lies in modeling the overlapping community search.* CS and OCS have different semantics. For OCS, each community has a "functioning model". For example, a person may be part of a "weekend hiking" community and a "software engineer community" at the same time. If we mix the two communities, then we incur a loss of semantics. Typical CS approaches cannot tell the difference. For example, one approach finds the subgraph that has the largest minimal degree as the community [3]. Clearly, if two subgraphs satisfy the criterion, then their union also satisfies the criterion. This shows that CS is not a good model for overlapping community. In comparison, OCS is more semantically aware in the sense that its goal is more about finding a functioning module that can better reveal the role of a vertex. Hence, we need to define the community more carefully and precisely to reflect this requirement. Specifically, we need to prescribe when two overlapping groups can be merged into a single community, and when they are two separate communities.

- *Our second challenge is that OCS is computationally harder than CS.* Intuitively, in OCS, all communities need to be enumerated. In some extreme cases, when there are an exponential number of valid communities, the enumeration is quite costly. Even if there is only a small number of valid communities, finding them among an exponential number of candidates is still a great challenge. In contrast, in most CS [3], for a given query vertex, there is at most one community to detect.

- *Our third challenge is to adopt approximate approaches so that we can scale to large graphs with millions of nodes.* Due to its computation intractability, it is difficult to scale up to million-node real life graphs. Finding the approximate results without compromising too much the quality of the solution (particularly the semantics of the community) is a challenge.

1.4 Contributions and Organization

In this paper, we propose a novel model for OCS, the theory based on the model, and the algorithms that solve OCS. More specifically, we make the following contributions:

- We introduce a model for OCS that satisfies not only the traditional community semantics, but also the semantics of overlapping. Furthermore, the two types of semantics are consistent in our model.

- We propose several algorithms to solve OCS. These algorithms produce meaningful results, and can handle million-node graphs.

- We offer the theoretical guidelines for parameter selection. The guidelines take into consideration the structural properties of real networks, and ensure the discovery of meaningful overlapping communities in real life complex networks.

- We conduct extensive experiments on real networks to justify our algorithms and theories. We also show many applications of OCS model and algorithms.

The rest of the paper is organized as follows. Section 2 discusses related work. Section 3 focuses on the model and the problem of overlapping community search. Section 4 describes a naive algorithm and an improved algorithm. We further propose a more efficient approximate algorithm. Experimental results in Section 5 show the performance and effectiveness of both the exact and the approximate algorithms. We show some potential applications for OCS model in Section 6. We conclude in Section 7.

2. RELATED WORK

Our work is closely related to work on several topics, including *overlapping community detection*, *dense subgraph discovery*, and *clique finding*.

Overlapping community detection. Palla et al. [1] first noticed that most real networks have overlapping community structures and proposed a percolation based method to detect overlapping communities in real networks. They also defined a community as a k-clique component. But they did not generalize it to allow relaxation. They developed a software called CFinder [4] based on the method to detect overlapping communities on biology networks. Their work was further extended with the EAGLE [5] algorithm that merges two cliques into one community if their Jaccard-similarity score is large enough. Gregory et al. [6] transformed the problem of overlapping community detection (OCD) on graph G into an equivalent non-overlapping community detection problem on another graph G', which is produced by duplicating certain vertices or edges in G. They used betweenness as the heuristic to select the vertex or edge to be duplicated. They further improved its performance by using local betweenness in a way that can be quickly computed to approximate the betweenness [7]. Ahn et al. [8] reinvented a community as a group of links instead of vertices so that they can always produce a hierarchical dendrogram on vertex-overlapped networks. Baumes et al. [9] proposed a greedy solution to grow overlapping communities. The algorithm first ranks all nodes according to some criterion, such as PageRank. Then, each node is added in turn into existing communities whose densities will increase after the acceptance of the new node as a member. Xu et al. [10] found that some vertices can not be classified into any communities. These vertices, called *hubs*, usually bridge different communities. So they proposed a linear algorithm, called SCAN, to find hubs and communities. Recently some efficient overlapping community detection algorithms were proposed, including SLPA [11], GCE [12] and OSLOM [13]. SLPA simulates the human communication behavior and realizes it into a speaker-listener label propagation algorithm. GCE identifies distinct cliques as seeds and expands these seeds by greedily optimizing a local fitness function. OSLOM finds overlapping communities by local optimization of a fitness function expressing the statistical significance of clusters with respect to random fluctuations.

Compared to OCD, OCS is a more light-weight problem model, and it is suitable for online query answering. By definition, OCD inherently implies intractable computation complexity. For example, CFinder needs to enumerate all k-cliques in a graph and then combine them in terms of overlap between cliques. Both clique enumeration and pairwise computation of clique similarity are costly. In contrast, in OCS, most resulting communities tend to occur around the local neighborhoods of the query vertex, allowing finding the answers by only exploring a part of the graph instead of processing the entire graph.

Dense subgraph discovery. In many graphs whose dense subgraphs have clear boundaries and can be separated from each other, communities are dense subgraphs. In this sense, our work is closely related to dense subgraph discovery. The problem of finding the densest subgraph is NP-hard, where the density of a graph $G(V, E)$ is usually measured by $\frac{2|E|}{|V|(|V|-1)}$. Doron et al. [14] gave an approximate algorithm for finding the dense k-vertex subgraphs of a given graph, with approximation ratio $O(n^{\delta})$ for some $\delta < \frac{1}{3}$. For practical use, Gibson et al. [15] used recursive shingling to find dense subgraphs on massive graphs. Their algorithms are scalable to web-scale graph with 50M hosts and 11B edges but without guarantees for the density of the result. To handle Internet-sized graph data in the form of stream, Bahman et al. [16] proposed a greedy solution for graph streams, which takes $O(log_{1+\epsilon}n)$ passes for any $\epsilon \geq 0$ yielding approximation factor of $2(1+\epsilon)$. Some other methods propose different definitions of dense subgraphs. For example, DN-graph is proposed in [17] and accompanied with a triangle-counting based solution. Although dense subgraph discovery is related to community discovery, they are essentially different from each other. A community is far beyond a dense subgraph. As a community, the density within the subgraph is expected to be significantly larger than the outside world of the subgraph. Furthermore, our problem is a typical search problem, in which a query vertex needs to be specified.

Clique finding. k-cliques underlie our community definition. There are already many k-clique related literatures. Most of them concern clique finding, i.e., finding a maximal clique or finding cliques with a size constraint. In general, the decision version of clique finding is NP-Complete [18]. Derényi [19] gave the threshold of edge-linking probability for the emergence of a giant k-clique component in an Erdos Renyi(ER) [20] random graph. To solve the problem practically, local search [21] is widely employed in many efficient solutions [22, 23, 24]. Along this direction, reactive local search (RLS) [22], dynamic local search (DLS) [23] and an improved DLS [24] have been proposed in turn. For disk-resident massive graphs, Cheng et al. [25] proposed an external-memory algorithm ExtMCE to reduce memory usage. They built a summary of small-sized graph, called H*-graph, to precisely encode the neighborhood information in the original graph. We only use k-clique to define community. In our problems, we not only need to find cliques around the query vertex, we also need to take care of clique adjacency. The second issue is not addressed in the clique-related work.

3. OVERLAPPING COMMUNITY

In the following, we first define the model as well as the problem (Section 3.1). Then, we justify the model by highlighting some of its important properties (Section 3.2). Finally, we discuss how to tune the parameters in the model to produce meaningful results (Section 3.3).

3.1 Model and Problem Definition

We start with the definition of OCS problem. Then, we generalize it into the (α, γ)-OCS problem.

3.1.1 OCS

Network communities are usually modeled by k-cliques [1]. Given an original network G, we derive a *k-clique graph* from G. In the k-clique graph, each node represents a k-clique in G, and an edge between two nodes means that the k-cliques the two nodes represent are adjacent in G, where adjacency is defined as follows:

DEFINITION 1 (CLIQUE ADJACENCY). *Two k-cliques are adjacent if they share $k-1$ vertices.*

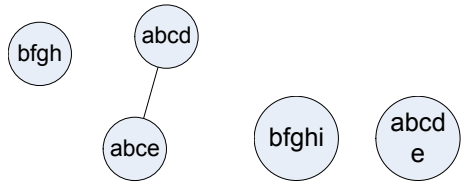
We define OCS based on the concept of k-clique components.

DEFINITION 2 (k-CLIQUE COMPONENT). *Let C be a connected component in the k-clique graph. A k-clique component is the union of all k-cliques represented by nodes in C.*

Intuitively, given a vertex v_0, there may be multiple communities that contain v_0. All these communities naturally constitute the overlapping communities that contain v_0. Thus, the problem is finding all such communities.

PROBLEM 1 (OVERLAPPING COMMUNITY SEARCH (OCS)). *Given a graph $G(V, E)$, a query vertex v_0 and a positive integer k, we need to find all k-clique components containing v_0.*

EXAMPLE 2 (OCS). *Consider the graph shown in Figure 1. Assume we want to find overlapping communities that contain node b (i.e., $v_0 = b$). Let $k = 4$. We have three k-cliques containing b: $C_1 = abcd$, $C_2 = abce$ and $C_3 = bfhg$. The clique graph is*



(a) The clique graph for $(3, 1)$-OCS (b) The clique graph for $(4, .8)$-OCS

Figure 2: Clique graphs are sensitive to the parameters of OCS.

shown in Figure 2(a). Hence, $C_1 \cup C_2$ and C_3 are returned as two overlapping communities containing the query vertex.

3.1.2 (α, γ)-OCS

The definition of overlapping community given above is too restrictive: First, every pair of nodes in each k-clique by definition must be connected. Second, two k-cliques are considered adjacent iff they share as many as $k-1$ nodes. In this section, we define a more general problem, (α, γ)-OCS, by relaxing the two constraints.

First, we relax the adjacency requirement. In (α, γ)-OCS, two k-cliques are considered adjacent if they share $\alpha \leq k-1$ vertices. This leads to the definition of α-adjacency.

DEFINITION 3 (α-ADJACENCY). *Two subgraphs G_1 and G_2 of graph $G(V, E)$ are α-adjacent if they share at least α vertices.*

Second, we relax the requirement of cliqueness. A k-clique is the densest graph among all k-node graphs. We generalize it to γ-quasi-clique by exposing parameter γ to tune the density of the subgraph.

DEFINITION 4 (γ-QUASI-k-CLIQUE [26]). *A γ-quasi-k-clique of graph G is a k-node subgraph of G with at least $\lfloor \gamma \frac{k(k-1)}{2} \rfloor$ edges (where $0 \leq \gamma \leq 1$).*

We now give a generalized version of overlapping community search: (α, γ)-OCS. Similar to the basic version, we also intend to find all overlapping communities containing a query vertex. But each community here is a γ-quasi-k-clique component and the clique adjacency is defined by α adjacency. Clearly, Problem 1 is simply $(k-1, 1)$-OCS.

PROBLEM 2 ((α, γ)-OCS). *For a graph G, a query vertex v_0 and a positive integer k, the (α, γ)-OCS problem finds all γ-quasi-k-clique components containing v_0.*

EXAMPLE 3 ((α, γ)-OCS). *To continue with the previous example, consider $(4, 0.8)$-OCS and $k = 5$ and $v_0 = b$. $abcde$ and $bghif$ are two resulting communities. This can be directly obtained from the clique graph shown in Figure 2(b).*

3.1.3 Complexity

We next analyze the complexity of overlapping community search. First, we show that $(k-1, 1)$-OCS is NP-hard. Given a query vertex v_0, the decision version of $(k-1, 1)$-OCS is to decide whether there are any k-cliques that contain v_0. When k is not a constant, this decision problem is NP-complete, which can be reduced from the k-clique problem (deciding whether a graph G has a k-clique). Then, as a more general problem, (α, γ)-OCS is also NP-hard.

3.2 The Rationale

Is the model proposed in Section 3.1 meaningful? In this section, we justify it from the following three aspects: community density, overlapping-awareness and consistency.

3.2.1 Community Density

To qualify as a community, a subgraph obviously needs to have *enough density*. In (α, γ)-OCS, the parameters α and γ together control the closeness among the members of a community. In this subsection, we establish a lower bound (Theorem 1) of the average degree for communities found by (α, γ)-OCS. Theorem 1 is based on Lemma 1. As an example of Theorem 1, the average degree of a community of $(k-1, 1)$-OCS is at least $k-1$.

LEMMA 1. *For a γ-quasi-k-clique, the number of edges incident to an arbitrary set of x vertices is at least $max\{0, \gamma\binom{k}{2} - \binom{k-x}{2}\}$.*

PROOF. The total number of edges of this clique is at least $\gamma\binom{k}{2}$. The subgraph induced by the rest $k-x$ vertices has at most $\binom{k-x}{2}$ edges, which are not incident to the selected vertices. Therefore, the lemma holds. □

THEOREM 1 (LOWER BOUND ON AVERAGE DEGREE). *The average degree of each community found by (α, γ)-OCS has a lower bound*

$$2 \max\{0, \min\{f(1), f(\alpha), f(k)\}\}$$

where $f(x) = \frac{\gamma\binom{k}{2} - \binom{k-x}{2}}{x}$.

PROOF. First, we show that for $x \in \{1, 2, ..., \alpha, k\}$, it holds that $f(x) \geq \min\{f(1), f(\alpha), f(k)\}$. We have $f'(x) = -\frac{1}{2} + \frac{(1-\gamma)k(k-1)}{2x^2}$, which satisfies the following inequalities:

$$\begin{cases} f'(x) \geq 0, 0 < x \leq \sqrt{(1-\gamma)k(k-1)} \\ f'(x) < 0, x > \sqrt{(1-\gamma)k(k-1)} \end{cases}$$

Thus, $f(x)$ takes its minimum value at $x = 1$, $x = \alpha$ or $x = k$, which means $f(x) \geq \min\{f(1), f(\alpha), f(k)\}$.

Now, suppose C is a community found by (α, γ)-OCS, and C consists of t different cliques $C_1, C_2, ..., C_t$. Let $V_i = \bigcup_{1 \leq j \leq i} C_j$. Let $v_i = |V_i - V_{i-1}|$, we have $|C| = \sum_{1 \leq i \leq t} v_i$. Let e_i be the number of edges in $E[V_i] - E[V_{i-1}]$, where $E[V_i]$ is the edges induced by V_i. The average degree of C is $\frac{2|E(C)|}{|V(C)|}$, which equals $\frac{\sum_{1 \leq i \leq t} 2e_i}{|C|} = \frac{2}{|C|} \sum_{1 \leq i \leq t, v_i \neq 0} v_i \times \frac{e_i}{v_i}$. Using the minimal value of $\frac{e_i}{v_i}$ to replace each $\frac{e_i}{v_i}$, we get

$$\frac{2|E(C)|}{|V(C)|} \geq \frac{2\sum_{1 \leq i \leq t, v_i \neq 0} v_i}{|C|} \min_{1 \leq i \leq t, v_i \neq 0} \{\frac{e_i}{v_i}\} = 2 \min_{1 \leq i \leq t, v_i \neq 0} \{\frac{e_i}{v_i}\}$$

Due to the definition, $v_1 = k, \forall 2 \leq i \leq t, v_i \leq \alpha$. According to Lemma 1 and the fact that $f(x) \geq \min\{f(1), f(\alpha), f(k)\}$, we find that the average degree of a community found by (α, γ)-OCS has a lower bound $2 \max\{0, \min\{f(1), f(\alpha), f(k)\}\}$. □

3.2.2 Overlapping Awareness

The model should be able to reveal the overlapping relationships of multiple communities. Note that this is not a trivial task. When two subgraphs overlap, it is hard to tell whether they are two components of a single community or two overlapping communities. In OCS, given two subgraphs H_1, H_2 that contain the query vertex, we have two choices: return $H_1 \cup H_2$ as a single community or return H_1, H_2 as two overlapping communities.

Previous models of communities were overlapping-unaware, and even overlapping-unfriendly. A widely adopted model [3] considers a subgraph whose minimal degree is larger than a given threshold as a valid community. Thus, the union of two communities is also a valid community. Hence, two communities can always be merged, even if they have very little overlap.

In (α, γ)-OCS, we use α to decide whether two components have enough overlapping to be considered and merged as a single community. Specifically, if two γ-quasi-k-cliques share fewer than α vertices and they are not reachable in any clique component, they are considered as two overlapping communities instead of a single community.

3.2.3 Consistency

Community search must be consistent, that is, if a community C is considered a community for a query vertex v_0, then if we use any other vertex in C as a query vertex, we should also obtain C as its community. We establish the consistency of (α, γ)-OCS in Theorem 2, which can be easily derived according to the fact that clique adjacency is an equivalence relationship.

THEOREM 2 (CONSISTENCY). *In (α, γ)-OCS, if C is a community that contains query vertex v_0, then for any other vertex $v \in C$ as the query vertex, C is also returned as its community.*

3.3 Parameter Selection

The model for overlapping community contains 3 parameters: k, α, γ. The selection of these parameters is crucial to producing meaningful results. In this section, we will give the theoretic guideline for selecting appropriate parameters.

3.3.1 Selection of γ and k

Parameter γ controls the density of the community. The selection of k and γ is related, in the sense that the larger the k, the more difficult it is to find communities with the high density. Hence, in order to generate meaningful results, γ should vary with k. Next, we will reveal the relationship between γ and k. We first give a simple relationship to avoid a trivial case. Then, we establish a more complicated relationship between them on ER random graphs.

A simple relationship. In general, as a meaningful community, the quasi clique needs to be connected. For k vertices, even when there are $\frac{k(k-1)}{2} - (k-1)$ edges among them, it is still possible that they are disconnected. It happens when the k-vertex graph consists of an isolated vertex and a $(k-1)$-vertex clique. To avoid the disconnected case, γ should be big enough. Hence, we need to ensure that:

$$\lfloor \gamma \frac{k(k-1)}{2} \rfloor > \frac{k(k-1)}{2} - k + 1,$$

By transformation, we have

$$\gamma \geq \frac{k-2}{k} + \frac{2}{k(k-1)} \tag{1}$$

Relationship on ER random graphs. Given a graph with N nodes and M edges, the density of the graph is given by $\frac{2M}{N(N-1)}$. Next, we use the ER random graph model [20] as an example to reveal the relationship between γ and k. The ER random graph is used to simulate real graphs, such as road networks. Consider an ER random graph $G \in \mathcal{G}(n, p)$ that has n vertices and each pair of vertices is linked with probability p, independent of other vertices and edges. Suppose we randomly pick k vertices from G. Let X be the random variable that represents the number of edges among the selected vertices. Then, the probability that $X = i$ is

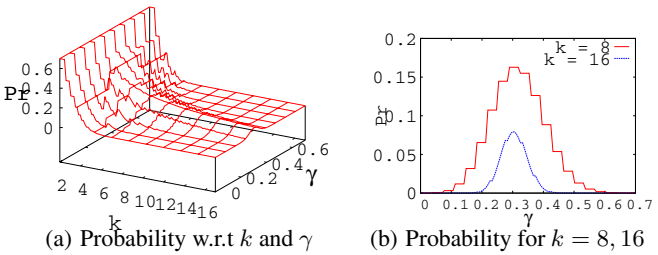
$$Pr(X = i) = \binom{m}{i} p^i (1-p)^{m-i} \tag{2}$$

where $m = k(k-1)/2$. Let Y be the random variable that represents the density of the subgraph induced by the randomly selected k vertices. Clearly, we have $Y = X/m$. Then, the probability that k selected vertices have density γ is a function of k and γ, that is:

$$Pr(Y = \gamma) = \binom{\frac{k(k-1)}{2}}{\lfloor \frac{\gamma k(k-1)}{2} \rfloor} p^{\lfloor \frac{\gamma k(k-1)}{2} \rfloor} (1-p)^{\frac{k(k-1)}{2} - \lfloor \frac{\gamma k(k-1)}{2} \rfloor} \tag{3}$$

We give the simulation of $Pr(Y = \gamma)$ in Figure 3(a) for $p = 0.3$. We can clearly see that when γ is fixed, $Pr(Y = \gamma)$ in general is a decreasing function of k. This suggests that we can find communities for small k with high probability. We justify this in the experimental section by showing that for most real networks communities exist mostly for $4 \leq k \leq 10$.

When k is fixed, $Pr(Y = \gamma)$ reaches its maximum when γ is close to the average density. To show this more clearly, we give $Pr(Y = \gamma)$ as a function of γ for two fixed k in Figure 3(b). The simulation results imply that there are many subgraphs of average density in the networks. In general, a meaningful community should have a much larger density than the average density. Hence, we should select a large enough density to disfavor those trivial communities with average density.



(a) Probability w.r.t k and γ (b) Probability for $k = 8, 16$

Figure 3: Simulation of density. This simulation shows the correlation between k and γ.

More formally, once k is given, we may use a confidence value such as $c = 0.05$ to construct an inequality $Pr(Y \geq \gamma) \leq c$ which is equivalent to

$$\sum_{i \leq j \leq m} \binom{m}{j} p^j (1-p)^{m-j} \leq c$$

, since $Pr(X \geq i) = Pr(Y \geq \gamma)$ when $i = \gamma m$. By solving above inequation, we find the appropriate i value and corresponding γ. By tuning c, we can control γ. A smaller c generally leads to a larger γ.

3.3.2 Selection of α

Parameter α, as we mentioned before, controls how strictly we consider two communities to be one community or components of the same community. Our major goal in selecting a meaningful α is to avoid two trivial cases.

The first trivial case occurs when the density of a γ-quasi-k-clique is contributed mainly by its α-size subset. As a result, any other $k - \alpha$ vertices outside of the clique combined with the subset will be a valid adjacent clique. This trivial case obviously leads to meaningless communities. To avoid this case, we need $\binom{\alpha}{2} < \gamma \binom{k}{2}$. By simple transformation, we have

$$\alpha \leq \lceil \sqrt{1/4 + k(k-1)\gamma} - 1/2 \rceil \tag{4}$$

The second trivial case occurs when two γ-quasi-k-cliques intersect at a α-size subset that is quite sparse. In real networks, if the common part of two subgraphs are too sparse, they can rarely be regarded as a single closely-connected community. If the common part of two subgraphs is too sparse, the two subgraphs can rarely be regarded as a single closely-connected community. This case happens if all edges are located among the rest $k - \alpha$ vertices. To avoid this, we set $\gamma \binom{k}{2} > \binom{k-\alpha}{2}$ so that the common part has at least one shared edge. By transformation, we have

$$\alpha \geq \lfloor k + 1/2 - \sqrt{\gamma k^2 - \gamma k + 1/4} \rfloor \tag{5}$$

Combining the two cases, given k and γ, we chose α to be an integer between the lower and upper bounds. As an example, when

$\gamma = 1$, we have $1 \leq \alpha \leq \lceil\sqrt{1/4 + k(k-1)} - 1/2\rceil$. Furthermore, when $k = 4$, we have $1 \leq \alpha \leq 3$. Hence, usually we use $\alpha = 3$.

3.3.3 Practical selection of parameters

Based on the above analysis, we show that $\alpha = k - 1$, $\gamma = 1$ (or values close to 1) are typical settings of (α, γ)-OCS. By simple transformation of Eq. 1, we have

$$\gamma \geq 1 - \frac{2}{k}\frac{k-2}{k-1} = f(k)$$

. When $k > 3$, $f(k)$ converges to 1. It can be observed from the simulation of $f(k)$ in Figure 4. Hence, $\gamma = 1$ (or values close to 1) is a typical setting. Next, we show that $\alpha = k - 1$ is also a typical setting because *only* $\alpha = k - 1$ lies between the lower and upper bound of α for any k and $\frac{k-2}{k} + \frac{2}{k(k-1)} \leq \gamma \leq 1$ (shown in Theorem 3).

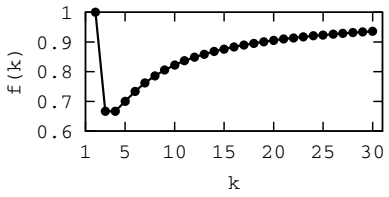


Figure 4: Simulation of $f(k)$.

THEOREM 3. *Let $\alpha = k - \Delta$ and Δ be a positive integer less than k. Only when $\Delta = 1$, α satisfies Eq. 4 and Eq. 5 for any $k \geq 2$ and $\frac{k-2}{k} + \frac{2}{k(k-1)} \leq \gamma \leq 1$.*

PROOF. First, consider $k = 2$. Since $0 < \alpha = k - \Delta < k$, we have $2 - \Delta > 0$. Hence, only $\Delta = 1$ satisfies this inequality. Next, for any k and $\frac{k-2}{k} + \frac{2}{k(k-1)} \leq \gamma \leq 1$, we have

$$\lceil\sqrt{1/4 + k(k-1)\gamma} - 1/2\rceil \geq \lceil\sqrt{9/4 + (k-1)(k-2)} - 1/2\rceil$$

, which is not less than $k - 1$; and

$$\lfloor k{+}1/2{-}\sqrt{\gamma k^2 - \gamma k + 1/4}\rfloor \leq \lfloor k{+}1/2{-}\sqrt{(k-1)(k-2) + 9/4}\rfloor$$

, which is not larger than $k - 1$. Thus only $\alpha = k - 1$, i.e., $\Delta = 1$, satisfies the constraints. □

4. ALGORITHMS

In this section, we will first propose two exact algorithms to solve OCS. The first is a straightforward solution. The second is a more efficient exact solution. To further improve the performance, we also propose a more efficient approximate algorithm.

4.1 Exact Algorithm

4.1.1 Naive Algorithm

We give a naive algorithm according to the definition of OCS. The direct solution for OCS consists of three major steps, which are shown in Algorithm 1. In the first step, we find the γ-quasi-k-clique that contains v_0. Then for each newly discovered vertex, we find all γ-quasi-k-cliques that contain the vertex. We repeat the above finding procedure until no new vertex can be discovered. In the second step, we calculate the clique adjacency for all cliques found in the first step. Finally, we return all clique components as the resulting communities.

4.1.2 New Algorithm Framework

The naive algorithm is not smart yet. As illustrated in Example 4, many cliques enumerated in the naive algorithm will not belong to a valid community. Hence, enumerating these invalid cliques

Algorithm 1 Naive OCS

Input: $G(V, E)$, v_0 α, γ, k;
Output: The overlapping communities containing v_0
//Stage 1. Find all the candidate cliques
1: $V_c \leftarrow \{v_0\}$
2: **for all** unvisited vertex $v \in V_C$ **do**
3: $find\_clique(v)$; ▷ find *all* cliques that contain v
4: Add all vertices of these cliques into V_C;
5: **end for**
//Stage 2. Calculate clique component
6: Calculate the adjacency matrix of candidate cliques;
//Stage 3. Return all clique components as communities

is wasteful. This can be avoided if we check the adjacency when a clique is enumerated. Following this idea, we proposed an improved algorithm, shown in Algorithm 2. The algorithm runs iteratively. In each iteration, we find an unvisited clique containing the query vertex by $next\_clique()$. If such a clique exists, we find the clique component that the clique belongs to by $expand()$.

EXAMPLE 4 (WASTEFUL ENUMERATION). *Consider $(3, 1)$-OCS with $k = 4$ and $v_0 = d$ on the graph shown in Figure 2(a), $\{a, b, c, d, e\}$ is the unique resulting community. By the naive algorithm, $bfgh$ will also be enumerated and participate in the succeeding computation, which is wasteful.*

Algorithm 2 Improved OCS

Input: $G(V, E)$, v_0 α, γ, k;
Output: The overlapping communities containing v_0
1: $\mathcal{R} \leftarrow \emptyset$;
2: **while** $C \leftarrow next\_clique(v_0), C \neq \emptyset$ **do**
3: $\mathbf{C} \leftarrow expand(C)$; ▷ Find the clique component of C
4: $\mathcal{R} \leftarrow \mathcal{R} \cup \{\mathbf{C}\}$;
5: **end while**
6: Return $\mathcal{R}$;

Note that for any two adjacent cliques, their clique components are identical. To avoid redundant enumeration, we record the visit status of each clique in both $next\_clique()$ and $expand()$, and only enumerate unvisited cliques. We illustrate this in Example 5. The new algorithm framework also allows us to further optimize it. We will present such optimizations in the following sections

EXAMPLE 5 (IMPROVED ALGORITHM). *Continue with the previous example but change v_0 to b. $next\_clique()$ may first return $bfgh$. By calling $expand()$ on this clique, we find the first community $\{b, f, g, h\}$. The next call of $next\_clique()$ may return $abce$. By calling $expand()$ on this clique, we find $abcd$. Thus, $\{a, b, c, d, e\}$ is the second community. Then, any further call of $next\_clique()$ will return nothing and the algorithm will terminate since all cliques have been visited either in $next\_clique()$ or $expand()$.*

4.1.3 Implementation of $next\_clique()$

$next\_clique(v_0)$ is responsible for enumerating each unvisited γ-quasi-k-clique containing vertex v_0. Finding all cliques is computationally hard. A naive solution needs to exhaustively enumerate all subsets with size k containing v_0, and check whether it is a valid clique. The solution costs $O(|V|^k k^2)$ time. Because there are $O(|V|^k)$ k-size subsets and checking whether the subset is a valid clique needs $O(k^2)$ time.

Clique enumeration is a well-known computationally hard problem. Even allowing approximation, clique finding is *hard to approximate*. Hence, brute-force enumeration seems to be inevitable to produce exact results. The major procedure is a depth-first search with backtracking. The procedure is shown in Algorithm 3. The search starts from $U = \{v_0\}$. Then, the procedure iteratively adds a new vertex into U until a new valid clique is found (line 5,6). The

procedure will stop until all γ-quasi-k-cliques containing the query vertex are found.

The backtracking search can be improved from two aspects. First, we just need to select a new vertex from the neighbors of the current vertex (line 13). This is because as a valid parameter, γ needs to satisfy Eq. 1, which ensures the result is a connected subgraph. Second, we speedup the search by pruning the impossible enumeration (line 11). Let $|E(U)|$ be the number of edges in the subgraph induced by U. Thus the maximal number of edges that the resulting clique has is $|E(U)| + (k - |U|)|U| + \frac{(k-|U|)(k-|U|-1)}{2}$, which equals $|E(U)| + \frac{(k-|U|)(k+|U|-1)}{2}$. When the maximal number of edges is less than $\gamma\frac{k(k-1)}{2}$, that is

$$g(U) = |E(U)| + \frac{(k-|U|)(k+|U|-1)}{2} < \gamma\frac{k(k-1)}{2} \quad (6)$$

, we can certainly prune the current U. The left side of the inequality is a function of U. We use $g(U)$ to denote it.

Algorithm 3 DFS procedure of $next\_clique(v_0)$

```
1: U ← {v_0};
2: DFS(U, v_0);
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:         end if
10:    end if
11:    if g(U) < γ k(k-1)/2 then Return;
12:    end if
13:    for all (u, v) ∈ E, v ∉ U do
14:        DFS(U ∪ {v}, v);
15:    end for
16: end procedure
```

4.1.4 Implementation of $expand()$

Function $expand(C)$ is used for finding the clique component containing C. By any graph traverse on the clique graph, we can certainly find the connected component. But the traverse orders are influential on the performance. Here, we use DFS order.

The procedure of $expand(C)$ is shown in Algorithm 4. The major framework is a DFS traverse on the clique graph. In the algorithm, A is a global variable storing the currently found vertices in the clique component. The key operation in the traverse is *finding a subset S_2 from C's neighborhoods to replace one of C's subset S_1 so that the new combination C' is a valid clique.* To keep the size of the clique as k, we enforce $|S_1| = |S_2|$ (line 8). We can also use Eq. 6 to terminate the search that will not lead to a valid clique (line 6).

The function $Candidate(C - S_1)$ is used for finding a subset S_2 with the same size as $|S_1|$ from the *local neighborhoods* of $C - S_1$. Note that we do not need to enumerate S_2 from the entire V. Because under the constraint of Eq. 1, the induced graph of $(C-S_1)\cup S_2$ is always connected. Hence, in the worst case, we just need to explore the $|S_1|$-*hop-neighborhood of* $C - S_1$, which contains all vertices that are at most $|S_1|$ hops away from any vertex in $C-S_1$. When k is small, the exploration of $|S_1|$-hop-neighborhood is effective.

4.1.5 Duplication detection

In both $next\_clique()$ and $expand()$ functions, we may meet the same clique from different paths. For example, for the clique abc. We may meet it by $a - b - c$ or $a - c - b$. Hence, we need to tell whether a clique has ever been visited. For this purpose, we use a hash table to store visited cliques, allowing in constant time

Algorithm 4 $expand(C)$

```
Input: A γ-quasi-k-clique C;
Output: The clique component of C
1: A ← C;
2: Expand_clique(C);
3: return A;

4: procedure EXPAND_CLIQUE(C)
5:     for all S_1 ⊂ C, |S_1| ≤ k − α do
6:         if g(C − S_1) < γ k(k−1)/2 then Continue
7:         end if
8:         for all S_2 ∈ Candidate(C − S_1), |S_1| = |S_2| do
9:             C' ← (C − S_1) ∪ S_2;
10:            if C' is unvisited And C' is a γ-quasi-k-clique then
11:                A ← A ∪ S_2;
12:                Expand_Clique(C');
13:            end if
14:        end for
15:    end for
16: end procedure
```

to query a visited clique. We use the following hash function:

$$h(C) = (\sum_{v_i \in C} id(v_i) \times a^{id(v_i)}) \mod b \quad (7)$$

, where a, b are two large primes and $id(v_i)$ is the id of v_i. Hashing each k-size clique takes $O(k)$ time. By DFS, such complexity can be further reduced to $O(1)$. We only show this on $(k-1,1)$-OCS. In this case, two successively visited cliques have only two different vertices in the DFS traverse order. Thus, we can calculate the hash value in an incremental way. Suppose, $C_{i+1} = (C_i - \{v\}) \cup \{u\}$, we have

$$h(C_{i+1}) = (h(C_i) - id(v) \times a^{id(v)} + id(u) \times a^{id(u)}) \mod b \quad (8)$$

4.1.6 Optimization on $(k-1,1)$-OCS

In the exact solutions for the general OCS problem, we can only detect duplication after the enumeration of a clique. For a k-size clique, we may enumerate it $O((k-1)!)$ times. Next we show that on $(k-1,1)$-OCS, we can avoid such redundant enumerations. We only discuss it for $next\_clique(v_0)$. $Expand()$ can be optimized with the same technique. When we search for a γ-quasi ($\gamma < 1$) clique containing v_0, we need exhaustive enumerations to find a clique (line 13-15 in Algorithm 3). However, on $(k-1,1)$-OCS, we are searching for a standard clique (a complete subgraph). It is *order independent* to find such cliques. That is, for these cliques we can use any DFS order to enumerate the same clique. As a result, on $(k-1,1)$-OCS, we can pose an arbitrary linear order on V. Then, we search for a clique according to the linear order. That is only adding a vertex with a superior order into the partial solution. In this way, redundant enumerations of a k-clique can be avoided.

4.2 Approximate Algorithm

Next, we propose an efficient approximate algorithm for (α,γ)-OCS. The algorithm shares the same framework as Algorithm 2 with $next\_clique()$ and $expand()$ further approximated.

4.2.1 Approximation in $next\_clique()$

The $next\_clique()$ function may enumerate an exponential number of γ-quasi-k-cliques. To reduce the enumeration space, we only enumerate an unvisited clique which contains at least one new vertex that does not belong to any communities already found. This heuristic can be achieved by a slight change of the DFS procedure (Algorithm 3). Let $v_0, v_1, .., v_k$ be the vertex sequence (staring from the root v_0) in the DFS order. We require that the second vertex v_1 is a new vertex. In this way, we ensure that the new clique contains at least a new vertex. Note that we enforce that the vertex sequence contains a new vertex as early as possible and v_1 is the

earliest vertex added into the sequence. Hence, v_1 is required to be a new vertex.

4.2.2 Approximation in $expand()$

We first study the traverse orders in $expand()$ function. We show in Example 6 that to meet all vertices in a community, the best case only requires a linear number cliques to be visited, while in the worst case, an exponential number of cliques may be visited.

EXAMPLE 6 (INFLUENCE OF TRAVERSE ORDERS). *Consider a complete graph with 100 vertices and $k = 4$. For $(k-1,1)$-OCS on any query vertex, the graph itself is the unique answer. To find the exact result, we need to visit $\binom{100}{4}$, approximately millions of k-cliques. Alternatively, if we always select a clique that is adjacent to previously visited cliques and contains at least one unvisited vertex, after visiting 97 cliques, all vertices in the result are met.*

The above example implies that we may find a community quite early before we exhaustively enumerate all γ-quasi-k-cliques in the community. Let C be a community of (α,γ)-OCS on graph G, in general we expect to find a shortest clique sequence $C_1, C_2, ...C_m$ such that each C_i is adjacent to at least one of its preceding cliques, and $\cup_{1\le i\le m} C_i = C$. However, in general, there exists an exponential number of possible sequences, it is hard to select the shortest sequence to visit.

Here, we will present a heuristic to find a short clique sequence. We still use DFS order. Consider $expand(C_1)$. Let C_i, C_{i+1} be two successively visited cliques; we always select C_{i+1} such that

$$C_{i+1} - \bigcup_{j\le i} C_j \neq \emptyset \tag{9}$$

. In other words, C_{i+1} is the clique containing at least one unvisited vertex. The traverse stops at C_m, to which no other unvisited cliques satisfying Eq. 9 are adjacent. In this way, the complete clique sequence $C_1, ...C_m$ is found. We return the union of them as the result of $expand(C_1)$. This heuristic can be implemented by simply adding a condition, $S_2 - A \neq \emptyset$, in line 10 of Algorithm 4.

4.2.3 Analysis

Consider a k-clique component C. Compared to the exact algorithm, which explores $O(\binom{|C|}{k})$ cliques, the approximate $expand()$ algorithm only explores exactly $O(|C|)$ cliques. Because the heuristic defined in Eq. 9 ensures finding a new vertex each time a clique is visited. $O(|C|)$ cliques will be visited and many wasteful enumerations are avoided. In this way, we reduce the exponential complexity to linear complexity. However, this heuristic may produce inexact results, which are illustrated in Example 7. We will show in the experimental section that this heuristic works quite well on real graphs, producing accurate results in most cases.

EXAMPLE 7 (BAD CASE). *Consider the $(2,1)$-OCS on the graph shown in Figure 1 with $k = 3$ and $v_0 = f$. The right community is $bfghi$. If cliques bfh, bgh were first visited, the search will stop since no new clique can be visited. As a result, i will be missed. If we first visit bfh and bfg, fgi is qualified to be visited and then we can find the right community.*

Due to the *NP-hardness to approximate* of clique finding, it is hard to give a theoretic guarantee of the approximation quality. However, we find that the approximate communities are *finer* than the exact ones. More formally, overlapping communities can be considered as a collection of subsets $P = \{C_1, C_2, ...C_k\}$. Then, for each community C_i in the approximate result, $\exists C'$ such that $C_i \subseteq C'$ and C' is an exact community of v_0. It clearly holds since the two approximations are equivalent to strengthen the constraint of a subgraph as a community. Since the community is finer, its size is smaller than or equal to the exact one. We have mentioned that the approximate algorithm only explores $O(|C|)$ cliques. Therefore, this property implies that the approximate algorithm is more efficient.

5. EXPERIMENTS

In this section, we present the experimental study and show the effectiveness and efficiency of our models and algorithms.

5.1 Experiment Setup

We ran all experiments on a PC with Intel Core2 at 2.13GHz, 4G memory running 64-bit Windows 7. We implemented all algorithms in C++. We compared the improved exact algorithm and the approximate algorithm. We also compared the OCS model and the OCD model. Except the study on the influence of parameters of γ and α, all results without explicit statement are obtained by $(k-1,1)$-OCS model.

We used four real networks to test our solution. The basic statistics of these networks are shown in Table 1. DBLP is a scientific collaboration network extracted from a recent snapshot of the DBLP dataset [2]. Each vertex in the graph represents an author and each edge indicates a coauthoring relationship. Livejournal [3] is the friendship network of Livejournal, a social networking and blogging site. Google represents the web graph that was released in 2002 by Google. In the graph, nodes represent web pages and directed edges represent hyperlinks between them. We ignore the direction of the edges. WordNet is a semantic network in which each vertex represent a specific sense in language and each edge represents a certain semantic relationship between senses. WordNet has been widely used in a variety of real applications, such as word sense disambiguation, automatic text classification and automatic text summarization.

5.2 Case Study

We justify the model by two case studies: the scientific collaboration network and WordNet.

5.2.1 Scientific collaboration network

We run both the improved exact algorithm and the approximate algorithm on the scientific collaboration network extracted from DBLP. The two algorithms produce almost the same results for most query vertices. Here, we use 'Jiawei Han' as the query vertex. Jiawei Han is a renowned computer scientist specializing in data mining and database. When $k = 6$, the exact result and the approximate result only differ in one community, where only two authors are missing in the approximate result. This justifies the effectiveness of the approximate algorithm. Hence, in the following evaluation, we will only use results produced by the approximate solution.

We show in Figure 5(a) part of the overlapping communities that Jiawei Han resides in. It only takes 18ms for the approximate solution to get the results, which justifies the efficiency of the approximate solution. Next, we focus on the effectiveness of the approach. We find that our overlapping community search model is meaningful in many aspects.

- First, it successfully unveils multiple research interests of an author. For example, in Jiawei Han's case, the community C_1 represents the topic of multimedia data mining and the community C_2 is about stream data mining. Each community has a clear boundary to be distinguished from others. We also can see that those authors with multiple research interests occur in different communities. For example, Jian Pei simultaneously occurs in C_2 and C_3, reflecting his research interest in both stream data mining and information network mining.

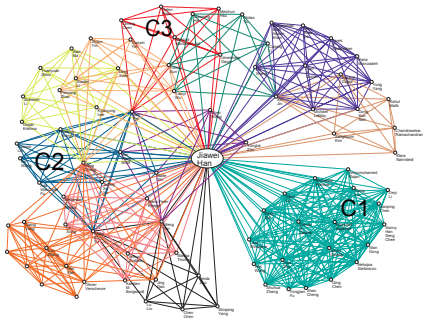
- Second, our model is flexible. By tuning parameter k, our model can discover communities with different closeness. For example, when $k = 9$, we discover more closely-connected communities of 'Jiawei Han' as shown in Figure 5(b). From

[2] Available at http://dblp.uni-trier.de/xml/

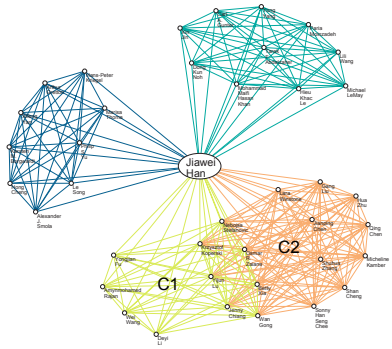
[3] This and Google are available at http://snap.stanford.edu/data.

Dataset	Basic information #Vertices	#Edges	average degree	Performance of OCD and OCS LA	Amortized LA	OSLOM2	Amortized OSLOM2	OCS
WordNet	82670	133445	3.23	51s	0.61ms	1913s	23.1ms	0.15ms (k=3)
DBLP	560851	1816613	6.48	6183s	11ms	6993s	12.5ms	6.24ms (k=4)
Google	916427	4322051	9.43	22873s	25ms	84725s	92.5ms	31ms (k=6)
Livejournal	4847571	42851237	17.7	$>= 24h$	N/A	N/A	N/A	64ms (k=9)

Table 1: Datasets, performance of OCD and OCS



(a) $k = 6$



(b) k=9

Figure 5: Case study on DBLP. OCS can find communities representing different research topics. By tuning k, OCS can find communities with different closeness. This case study justifies the effectiveness of the OCS model.

the result, we can see that community C_1 of $k = 6$ is broken up into two more-closely connected overlapping communities C_1, C_2 of $k = 9$. The two communities respectively reflect Jiawei's more specific research interest. One is about multimedia data mining and the other is about association rule mining on large relational databases. When k becomes larger, many periphery members of communities who have relatively fewer collaborations with members in the community are excluded from the community. For example, we find that 59 authors in the result when $k = 6$ are excluded from the result when $k = 9$.

5.2.2 *WordNet*

In WordNet, we use the word *'vessel'* as the query vertex and we set $k = 3$. The result in Figure 6 shows that *'vessel'* occurs in multiple communities, and each community represents a certain sense of *'vessel'*. The community that contains *'bowl'*, *'dish'* corresponds to tablewares. The community consisting of *'barrel'* and *'tube'* represents the meaning of bucket. Another two communities represent the meaning of ships. The difference between these two communities is that the one consisting of *'ship'* and *'galley'* is generally used for describing ships with large tonnage and the other community are usually used for describing small boats. This example shows that our OCS model is helpful for finding different senses of a word, which is a fundamental task in natural language understanding.

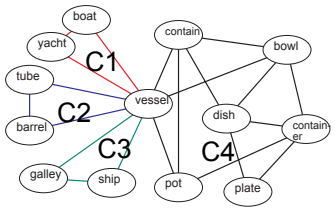


Figure 6: Case study on WordNet. OCS can find different senses of a word.

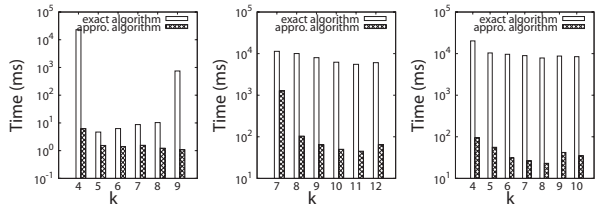
5.3 Performance

We first compare the performance of the exact algorithm and the approximate algorithm. Then, we compare the performance of OCS and OCD.

OCS: exact vs approximate methods. We tested the performance for both exact algorithms and approximate algorithms. For each k, we randomly selected 100 *valid query vertices* (with degree at least $k - 1$). We counted the average query answering time for these vertices under different k. Note that the exact solution has exponential enumeration cost. Hence, when the running time exceeds 60s, we terminated the exact algorithm and recorded the entire query response time as 60s. Clearly, this favors exact solutions. However, approximate solution still shows significant advantage over exact solutions as the following results indicate.

The results on three real networks are given in Figure 7. We can see that in general approximate algorithm is more than two orders of magnitudes faster than the exact one. Note that the performance priority of the approximation algorithm over the exact one is more striking than observed since we limit the running time of the exact algorithm to 60s. For Livejournal and Google, significant performance difference can be consistently observed over different k. When $k = 4$, most valid query vertices have non-trivial results and the approximate algorithm shows a much clearer advantage over the exact algorithm. On DBLP, an increase in speed by three orders of magnitudes is achieved for $k = 4$. For all the tested networks, the approximate solution's performance is quite stable compared to the exact solution, especially on DBLP. Even for the biggest two datasets, Livejournal and Google, both with approximately a million nodes and tens of millions edges, the approximation algorithm returns results within less than 100ms for almost all k. This sufficiently shows that the approximate solution can be used as online service for large real graphs.

OCS vs OCD. For comparisons, we also give the running time of the state-of-the-art OCD algorithmss LA [9] and OSLOM2 [13] in Table 1. We use the $-fast$ option to get the fastest results for OSLOM2. The time of OSLOM2 on Livejournal is not available. Because the program, which was downloaded from its official website, exited unexpectedly on Livejournal. We find that even on the smallest graph WordNet, two OCD algorithms still need 51s or 1913s, respectively. For a fair comparison, we also give the amortized running time of OCD, that is $\frac{\text{running time of OCD}}{|V|}$. Even con-



(a) DBLP (b) Livejournal (c) Google

Figure 7: Performance. The performance of the approximate solution is significantly better than that of the exact solution.

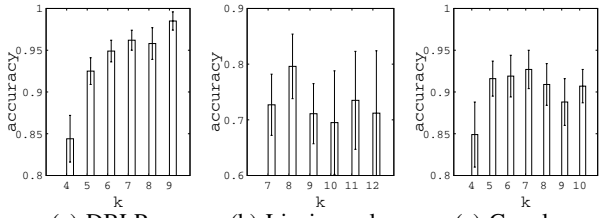
sidering the amortized running time, OCS is more efficient than OCD. From the comparison, we can clearly see that OCD is computationally prohibitive on large networks. In contrast, OCS is a lightweight solution to help us find overlapping communities.

5.4 Quality

In this subsection, we show the quality of the solution provided by the approximate algorithm. Overlapping communities express a certain equivalence relationship on V. Any two vertices in C_i are equivalent to each other under this equivalence relationship. Thus, we may use the overlapping ratio of the equivalence relationship to quantify the similarities between exact and inexact results. Let $P_1 = \{C_1, C_2, ...C_k\}$ be the approximate result, and let C_i' be one of the exact communities that contain C_i as a subset. Such C_i' certainly exists since P_1 is finer than the exact communities P_2. Then, we define the similarity between P_1 and P_2 as

$$sim(P_1, P_2) = \max_{1 \le i \le k} \sqrt{\frac{|C_i|(|C_i| - 1)}{|C_i'|(|C_i'| - 1)}} \tag{10}$$

. Clearly, we have $0 \le sim(P_1, P_2) \le 1$.



(a) DBLP (b) Livejournal (c) Google

Figure 8: Accuracy of approximate algorithm. Communities produced by approximate solutions are quite close to that produced by exact solutions.

We ran the approximate algorithm with 100 randomly selected valid query vertices on each network under different k. For each query vertex, we calculated the similarity value defined in Equation 10. We summarized the average and variance of the similarity value for 100 random query vertices. The results are shown in Figure 8. It is clear that for each tested network and each k, more than 70% accuracy can be achieved. In some special cases, for example, on DBLP and Google, almost 90% accuracy can be achieved.

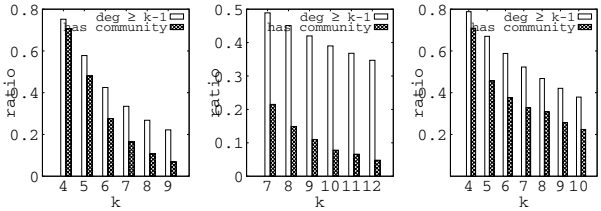
We give the detailed accuracy results of DBLP in Table 2. In the table, for each k, 100 randomly selected valid vertices were queried. $\#has\ community$ denotes the number of vertices among them that have valid communities. $\#equivalent$ is the number of vertices for which the approximate algorithm produces the exact results. And $ratio = \frac{\#equivalent}{\#has\ community}$. We can see that for any k, more than 79% of query vertices have the exact results. When k increases, the accuracy increases. In many real applications where community has vague meanings, the accuracy of the community is not a strict requirement and our approximate algorithm is quite suitable for these applications.

k	4	5	6	7	8	9
$\#equivalent$	66	69	58	43	37	29
$\#has\ community$	84	83	65	49	40	30
ratio	79%	83%	89%	88%	93%	97%

Table 2: Accuracy on DBLP. For most query vertices, the approximate solution produces exact results.

5.5 Effectiveness

To show the effectiveness of the OCS model, we computed the proportion of valid query vertices that have communities. We ran the approximate algorithm on 100 randomly selected valid query vertices. For comparison, the proportions of valid query vertices are also given. The results are shown in Figure 9. From the figure, we can see that for each network, OCS model can discover meaningful communities for a significant number of vertices. On both DBLP and Google, when $k = 4$, more than 90% of valid query vertices have communities. For all the networks, the ratios of valid vertices and those with communities decrease as k increases. This can be naturally explained, since the constraint on the community is more restricted for a larger k.



(a) DBLP (b) Livejournal (c) Google

Figure 9: Effectiveness of OCS. For most vertices, OCS model can find non-trivial results.

5.6 Influence of k

The value k is the most important parameter of OCS. In this subsection, we investigate the influence of k on the result of communities. We are interested in the number of communities as well as the size of communities.

We first give our analysis about the influence of k on community size and community number. In general, it is expected that the community size monotonically decreases with k. When k is small, the constraint for a group of vertices forming a community is weak. Consequently, it is easy to find large communities. When k grows, the community will become smaller. The influence of k on community number is a little bit more complicated. In general, there are two consequences caused by the growth of k:

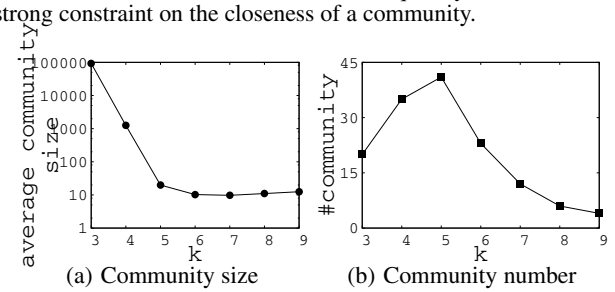
- (1)The constraint on the closeness of a community becomes stronger. As a result, many communities found under a small k will not be a valid community any more. Consequently, the number of communities will reduce when k grows.
- (2)The condition for two communities to be merged into one single community becomes stronger. As a result, large communities will be broken up into small communities. Consequently, the number of communities will increase with k.

Hence, it is quite possible that there is a critical point on which two forces are balanced and maximal number of communities can be achieved. To verify this conjecture, we studied communities of 'Jiawei Han' in DBLP. We summarized the community number and community size of 'Jiawei Han' as the function of k.

Table 3: Performance on WordNet

k	3	4					
α	2	2	2	2	3	3	3
γ	1	0.8	0.9	1	0.8	0.9	1
Time (ms)	2298	36044	18022	6000	17882	17801	3395

The results are shown in Figure 10 and they verify our conjecture. From Figure 10(a), we can clearly see that the community size monotonically decreases with k. When $k = 5$ and larger, the community size is almost stable. These communities in general are core-coauthoring team members of 'Jiawei Han'. From Figure 10(b), we can clearly see that $k = 5$ is the critical point on which the maximal number of communities are found. Before this, larger communities are broken up into smaller communities when k increases. After this, valid communities quickly vanish due to the strong constraint on the closeness of a community.



(a) Community size (b) Community number

Figure 10: Influence of k. Community size is monotonically decreasing with k.

5.7 Influence of α and γ

In this subsection, we study the influence of α and γ. We show our results on *WordNet*. Similar results were obtained on other networks. Since the graph is sparse, we only tested our approach on $k = 3$ and $k = 4$. According to Eq. 1, we have $\gamma \geq \frac{2}{3}$ for $k = 3, 4$. So we chose some representative γ satisfying these constraints. Note that for $k = 3$, only $\gamma = 1$ is meaningful. Hence, we only show the result for $\gamma = 1$ when $k = 3$. If $k = 4$, when $\frac{2}{3} \leq \gamma < \frac{5}{6}$, the quasi-4-clique has at least four edges; when $\frac{5}{6} \leq \gamma < 1$, the quasi-4-clique has at least five edges. We used $\gamma = 0.8$ and $\gamma = 0.9$ as two representative values of these two ranges, respectively. For each selected k and γ, we randomly selected 10 valid query vertices as v_0 and recorded their average running time in Table 3. When the query time of a vertex exceeds 60s, we considered it as 60s.

From the results, we found that, for the same k and α, generally a smaller γ costs more time. This is because a higher γ means a more strict constraint, which reduces the search space. For the same k and γ, the running time decreases with the growth of α. The reason is similar: a smaller α leads to a bigger search space.

6. APPLICATIONS

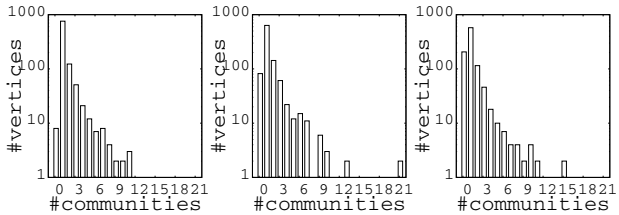
In this section, we propose two typical applications of OCS: *diversity based social network analysis* and *name disambiguation in social networks*. Our model and solution enable us to measure the diversity of social ties for a person, which is critical for the diversity based analysis in a quantitative manner. The OCS model and our solution also provide new insights to solve name disambiguation in social networks, which is well known as a hard problem. All the following studies are conducted on $(k-1, 1)$-OCS model.

6.1 Diversity-based Social Network Analysis

In social networks, an individual's social ties are diverse if he or she maintains connections to different communities or groups. Individuals with diverse social ties are more competitive than others [27, 28], because they serve as an intermediary with a position bridging different groups in a social network, and consequently own more opportunities than others in the network. Diversity leading to competition advantage has been widely acknowledged in real life, but has been rarely verified in a quantified manner. Computing overlapping communities is obviously a critical step to quantify the diversity of an individual. Due to the computational hardness of overlapping community detection, previous diversity measures use attributes to derive overlapping communities. For example, Shi et al. [29] uses the conferences at which an author has ever published a paper to obtain overlapping communities for DBLP scientific collaboration network.

Now, we can directly use our solution under the OCS model to calculate the diversity of an individual. Intuitively, the diversity of a person can be measured by the number of overlapping communities found by our approach. Next, we show some diversity related analysis on DBLP with diversity calculated by our OCS solution.

Diversity distribution. The first class of fundamental questions includes: *what is the distribution of diversity? Can we find people with really large diversity?* To answer these questions, we randomly select 1000 vertices and summarize the diversity distributions for $k = 3, 4, 5$. The results are shown in Figure 11. We find that most persons have diversity 1, indicating that they coauthor with authors in a local research community. These authors are mostly fresh hands in an area. However, there are some authors with large diversity. Some authors have diversity at about 20. These authors are mostly distinguished scientists in computer science. There are also some ambiguous names shared by more than one author. We will discuss this problem in the next application.



(a) $k = 3$ (b) $k = 4$ (c) $k = 5$

Figure 11: Distribution of diversity

Diversity leading to competitive advantage. Next, we verify the widely-established conjecture: *diversity of social ties leads to competitive advantage in a social network.* In a social network, a person's competitive power in general is positively correlated to his degree. Hence, to verify this conjecture, we summarize the correlation between degree and diversity. The result is shown in Figure 12. We find positive correlations between degree and diversity, which verifies the conjecture. This finding strongly suggests that diversity is one of the important driving forces for an author to become an academic star. This implication opens opportunity for some other interesting real applications, such as predicting rising stars.

6.2 Name Disambiguation

Name disambiguation is a typical problem in social network data management [30, 31]. For example, in DBLP there are at least 40 authors with the same name 'Wei Wang'. In general, name disambiguation is a challenging problem when person identity information is missing. The OCS model and its solution provide new insight to solve this problem. They are at least helpful in two aspects: First, they can be used to identify ambiguous names. Second, the overlapping communities themselves are candidate entities with the same name.

We ran our solution to $(k-1, 1)$-OCS with $k = 4$ on each vertex of DBLP. Table 4 presents the ambiguous names that have sig-

Ambiguous names			Renowned authors with large diversity		
Name	degree	# communities	Name	degree	# communities
Wei Li	589	113	Jennifer Widom	106	11
Xin Li	399	67	Alon Halevy	62	7
Ming Li	399	72	Hector Garcia-Molina	182	26
Li Zhang	486	95	David Dewitt	142	20
Ying Zhang	282	59	Michael Stonebraker	171	22
Hui Zhang	281	58	Jeffrey D. Ullman	97	11

Table 4: Ambiguous names, renowned authors in DBLP. Ambiguous names have a significantly large number of communities.

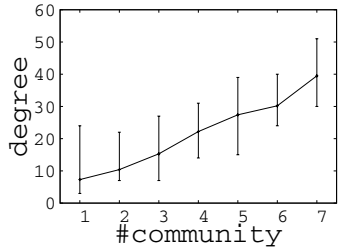


Figure 12: Correlation between diversity and degree (with $k = 4$). Diversity is correlated to degree.

nificant number of entities in DBLP. We found that most of these names have a large number of communities. Their community numbers in general can be considered as exceptions when compared to the community number distribution observed in Figure 11. For comparisons, we also give the community number of renowned scientists in the database community. It is clear that for a real person, even as famous scientists with many papers and abundant coauthoring relationships, their community numbers are smaller than the community number of ambiguous names. These facts strongly suggest that the large number of overlapping communities is a good indicator of ambiguous names referring to multiple entities. Besides name disambiguation in social networks, OCS is also helpful for sense disambiguation in semantic networks, as shown in the case study on WordNet.

7. CONCLUSION

Most real networks have overlapping community structures. In this paper we propose a novel overlappingness-aware community search problem: *overlapping community search*. Compared to overlapping community detection, our model is much lightweight and supports online query answering. We reveal the rationale behind the model and provide theoretical guidelines for tuning the model. We devise several exact algorithms and an efficient approximate algorithm to find meaningful overlapping communities. We conduct extensive experiments to show that both the exact algorithms and approximate algorithms are effective to discover meaningful overlapping communities in real networks, and the approximate solution is quite efficient, supporting online (within several ms) query answering on million-node graphs.

8. REFERENCES

[1] G. Palla, I. Derényi, I. Farkas, and T. Vicsek, "Uncovering the overlapping community structure of complex networks in nature and society." *Nature*, vol. 435, no. 7043, pp. 814–818, Jun. 2005.

[2] http://www.facebook.com/press/info.php?statistics.

[3] M. Sozio and A. Gionis, "The community-search problem and how to plan a successful cocktail party," in *KDD*, 2010.

[4] B. Adamcsek, G. Palla, I. J. Farkas, I. Derĺenyi, and T. Vicsek, "Cfinder: locating cliques and overlapping modules in biological networks," *Bioinformatics*, vol. 22, no. 8, pp. 1021–1023, 2006.

[5] H. Shen, X. Cheng, K. Cai, and M.-B. Hu, "Detect overlapping and hierarchical community structure in networks," *Physica A*, vol. 388, no. 8, pp. 1706 – 1712, 2009.

[6] S. Gregory, "An algorithm to find overlapping community structure in networks," in *PKDD*, 2007.

[7] ——, "A fast algorithm to find overlapping communities in networks," in *Machine Learning and Knowledge Discovery in Databases*, ser. Lecture Notes in Computer Science, 2008, vol. 5211, pp. 408–423.

[8] Y.-Y. Ahn, J. P. Bagrow, and S. Lehmann, "Link communities reveal multiscale complexity in networks," *Nature*, vol. 466, pp. 761–764, Jun. 2010.

[9] J. Baumes, M. Goldberg, and M. Magdon-ismail, "Efficient identification of overlapping communities," in *ISI*, 2005, pp. 27–36.

[10] X. Xu, N. Yuruk, Z. Feng, and T. A. Schweiger, "Scan: a structural clustering algorithm for networks," in *KDD*, 2007.

[11] J. Xie, B. K. Szymanski, and X. Liu, "Slpa: Uncovering overlapping communities in social networks via a speaker-listener interaction dynamic process," in *ICDMW*, 2011.

[12] C. Lee, F. Reid, A. McDaid, and N. Hurley, "Detecting highly overlapping community structure by greedy clique expansion," *arXiv preprint arXiv:1002.1827*, 2010.

[13] A. Lancichinetti, F. Radicchi, J. J. Ramasco, and S. Fortunato, "Finding statistically significant communities in networks," *PloS one*, vol. 6, no. 4, p. e18961, 2011.

[14] D. Goldstein and M. Langberg, "The dense k subgraph problem," *CoRR*, vol. abs/0912.5327, 2009.

[15] D. Gibson, R. Kumar, and A. Tomkins, "Discovering large dense subgraphs in massive graphs," in *VLDB*, 2005.

[16] S. V. Bahman Bahmani, Ravi Kumar, "Densest Subgraph in Streaming and MapReduce," in *PVLDB*, 2012.

[17] N. Wang, J. Zhang, K.-L. Tan, and A. K. H. Tung, "On triangulation-based dense neighborhood graphs discovery," in *PVLDB*, 2010.

[18] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations*, 1972, pp. 85–103.

[19] I. Derényi, G. Palla, and T. Vicsek, "Clique Percolation in Random Networks," *Physical Review Letters*, vol. 94, no. 16, pp. 160 202+, Apr. 2005.

[20] P. Erdos and A. Rényi, "On the evolution of random graphs," in *Publication of the Mathematical Institue of the Hungaria Academy of Sciences*, 1960, pp. 17–61.

[21] E. Aarts and J. K. Lenstra, Eds., *Local Search in Combinatorial Optimization*, 1997.

[22] R. Battiti and M. Protasi, "Reactive local search for the maximum clique problem," *Algorithmica*, vol. 29, no. 4, pp. 610–637, 2001.

[23] W. Pullan and H. H. Hoos, "Dynamic local search for the maximum clique problem," *J. Artif. Int. Res.*, vol. 25, pp. 159–185, February 2006.

[24] A. Grosso, M. Locatelli, and W. Pullan, "Simple ingredients leading to very efficient heuristics for the maximum clique problem," *Journal of Heuristics*, vol. 14, pp. 587–612, December 2008.

[25] J. Cheng, A. W.-c. Fu, and J. X. Yu, "Finding maximal cliques in massive networks by h*-graph," in *SIGMOD*, 2010.

[26] M. Brunato, H. Hoos, and R. Battiti, "On effectively finding maximal quasi-cliques in graphs," *Learning and Intelligent Optimization*, pp. 41–55, 2008.

[27] N. Eagle, M. Macy, and R. Claxton, "Network Diversity and Economic Development," *Science*, vol. 328, no. 5981, pp. 1029–1031, 2010.

[28] R. S. Burt, *Structural holes: The social structure of competition*. Harvard University Press, 1992.

[29] Q. Shi, B. Xu, X. Xu, Y. Xiao, W. Wang, and H. Wang, "Diversity of social ties in scientific collaboration networks," *Physica A*, vol. 390, no. 23?4, pp. 4627 – 4635, 2011.

[30] H. Han, H. Zha, and C. L. Giles, "Name disambiguation in author citations using a k-way spectral clustering method," in *ICDL*, 2005.

[31] H. Han, L. Giles, H. Zha, C. Li, and K. Tsioutsiouliklis, "Two supervised learning approaches for name disambiguation in author citations," in *JCDL*, 2004.