Query-Centralized Community Search

Zhuo Wang
Institute of Information Engineering, Chinese Academy of Sciences
School of Cyber Security, University of Chinese Academy of Sciences
wangzhuo@iie.ac.cn

Chaokun Wang
School of Software, Tsinghua University
chaokun@tsinghua.edu.cn

Weiping Wang
Institute of Information Engineering, Chinese Academy of Sciences
wangweiping@iie.ac.cn

Xiaoyan Gu
Institute of Information Engineering, Chinese Academy of Sciences
guxiaoyan@iie.ac.cn

Bo Li
Institute of Information Engineering, Chinese Academy of Sciences
libo@iie.ac.cn

Dan Meng
Institute of Information Engineering, Chinese Academy of Sciences
mengdan@iie.ac.cn

ABSTRACT

In this paper, we study the query-centralized version of community search: given a set of query nodes Q of a graph G, find a densely connected subgraph G_s where the query nodes play important roles. The task discovers communities largely influencing or influenced by Q and thus is valuable for the personalized recommendation. Existing models for community search are not suitable for the task as they may find a density maximized subgraph where the query nodes are marginalized. For this reason, we propose the KDR model which formulates a community as the subgraph with the maximized query-biased edge density among the connected k-core subgraphs containing Q with the largest $\frac{k}{d}$ (d denotes the query distance). We prove that finding a KDR-community is NP-hard and devise effective methods (KD and Loc-kdr) for the problems related to the KDR model. We further develop speed-up strategies to make Loc-kdr scalable to large networks. Experiments on real and synthetic networks demonstrate the performance of our solutions for finding KDR-communities.

CCS CONCEPTS

• **Information systems** → **Data mining**; • **Mathematics of computing** → **Graph algorithms**;

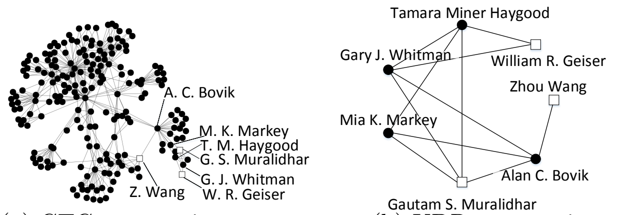
KEYWORDS

community search, graph algorithms, social networks

1 INTRODUCTION

With the rapidly increasing size of networks, the concept of community search has attracted great interests recently. It aims to find a "community" containing a set of given query nodes. The task is query-dependent. For example, the community formed by ***Alice*** and her family members differs from the one formed by her and her workmates [28]. Hence community search shows great potential in detecting personalized communities and has a wide range of applications such as team formation, tag suggestion and advertising [8, 28].

For community search, various effective methods [2, 17, 28, 30] have been proposed based on dense subgraph structures. However, the methods may find a density maximized subgraph of large diameter which is meaningless for the query nodes Q. As a case study, we build a co-author network of DBLP in which each edge between two authors indicates they have co-authored more than two times [17, 30]. Fig.1(a) shows the closest-truss-community [17]: (1) The query nodes are marginalized. (2) The members come from various backgrounds. (3) Many members (60%) in the community publish articles in NeuroComputing whereas none of the query nodes have articles published in the journal. In contrary, Fig.1(b) shows a community much more relevant to Q: (1) The query nodes are close to the other members and centred. (2) All the members have the experience studying or working at the University of Texas at Austin or the University of Texas MD Anderson Cancer Center. (3) Each member has articles published in J. Digital Imaging. Obviously, the community in Fig.1(b) is better as it is query-centralized and more meaningful to the query nodes.



(a) CTC-community (b) KDR-community

Figure 1: Communities discovered by CTC and KDR. We sample three nodes as a query and obtain the subgraphs of the DBLP co-author network. The query nodes are close to each other and publish articles in the same journals. The white rectangles are the query nodes and the black circles are the discovered members.

In this paper, we study query-centralized community search. Given a graph G and a set of query nodes Q, the task finds a densely connected subgraph G_s in which the query nodes play important roles. As the query-centralized community (such as Fig.1(b)) is meaningful to Q, our task is valuable for the personalized recommendation.

Our first contribution is devising a novel model for query-centralized community search. Based on k-core [27] and query distance d [17], we design the kd-metric ($\frac{k}{d}$) to recognize subgraphs of large k and small d. Furthermore, we devise query-biased relevance to shift the importance around Q. Our model is built on the kd-metric and relevance (Thus the model is called KDR). It requires a community to be a connected k-core subgraph G_s with the largest $\frac{k}{d}$ containing Q. Besides, the weighted edge density $\rho_w = \frac{\sum_{v \in G_s} deg(v) r_{G_s}(v)}{|V(G_s)|(|V(G_s)|-1)}$ should be maximized, where $r_{G_s}(v)$ is query-biased relevance of a vertex v in G_s. Fig.1(b) is an example of the KDR-community.

Our second contribution is providing methods for finding KDR-communities. We develop a method to discover $\frac{k}{d}$ maximized subgraphs. However, we find it NP-hard to discover a KDR-community. Hence we propose a heuristic method (Loc-kdr) which explores promising vertices around a Steiner tree iteratively until the metrics of the KDR model are large enough. By designing a novel distance function, the Steiner tree can be a good sketch for local exploration.

Our third contribution is accelerating connecting the query nodes. The demand for the connectivity of the query nodes makes the methods for community search inefficient. The problem also exists in our query-centralized version. Prior works [2, 17, 28, 30] in community search build a Steiner tree to connect the nodes. However, the usually adopted algorithms [21, 25] visiting the whole graph are computationally expensive in large graphs. Considering the feature that the query nodes in a community are close to each other, we propose two strategies to speed up building the Steiner tree in Loc-kdr. The first one is a pruning strategy with provable guarantees and the second one is a heuristic of best-first-search.

We extensively conduct experiments with various methods on real and synthetic networks containing ground-truth communities. The results show that our model and algorithms are effective and efficient in each network.

The rest of the paper is organized as follows. We define the KDR model and the corresponding problems in Section 2. In Section 3, algorithms for the KDR model and speed-up optimizations are proposed. Thereafter, the performance evaluation is presented in Section 4. We review the related work in Section 5 and draw our conclusion in Section 6.

2 MODEL & PROBLEM

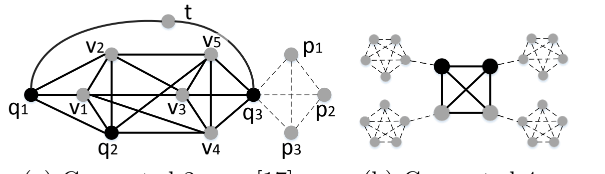
Our model and problems are defined based on an undirected, unweighted and connected graph $G(V, E)$, where V and E represent the set of vertices and edges respectively. Table 1 lists the commonly used notations in this paper. The query distance of a vertex v ($dist_G(v, Q)$) is the maximum length among all the shortest paths from v to any vertex $q \in Q$. Based on this, the query distance of a graph G is defined as $dist(G, Q) = max_{v \in G} dist_G(v, Q)$.

Definition 2.1. (CONNECTED K-CORE). Given a graph $G(V, E)$ and an integer k, a *connected k-core* is a connected subgraph $H \subseteq G$ where $\forall v \in V(H)$, $deg_H(v) \geq k$.

Prior models for community search maximize k based on k-core/truss structures [2, 17, 28]. Intuitively, maximizing k finds dense subgraphs. However, it will discover many irrelevant vertices if the latent community is not a k-core with the largest k. Given the graph $G(V, E)$ from Fig.2(b), we

Table 1: Main Symbols

Symbol	Definition
$G(V, E)$	A connected, undirected and simple graph
$Q; q$	A set of query nodes Q; a query node q
$N(v); deg(v)$	Neighbours/degree of v in a graph
$dist_G(u, v)$	Length of the shortest path from u to v in G
$dist_G(u, Q)$	Query distance of u, $max_{q \in Q} dist_G(u, q)$
$dist(G, Q)$	Query distance of G, $max_{v \in G} dist_G(v, Q)$
$\delta(.); f(.)$	The minimum degree/$\frac{k}{d}$ of a subgraph
$r_{G_s}(v)$	Query biased relevance of v in the subgraph G_s
$\rho_w(G_s)$	The weighted edge density $\frac{\sum_{v \in G_s} deg(v) r_{G_s}(v)}{|V(G_s)|(|V(G_s)|-1)}$



(a) Connected 2-core [17] (b) Connected 4-core

Figure 2: KD examples. In each figure, the subgraph induced by thick solid edges are $\frac{k}{d}$ maximized if the query consists of the black nodes.

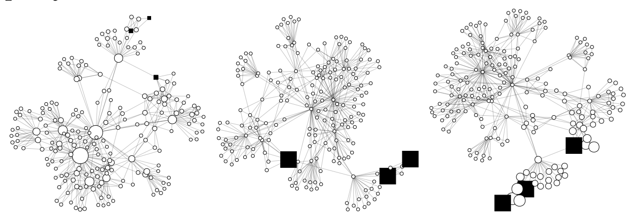

(a) PageRank (b) PHP (c) QBR

Figure 3: Relevance distribution of Fig.1(a) under different proximity measures. We use the max relevance of the vertices to normalize the vertex size. The black rectangles are the query nodes.

assume that a query Q consists of the black nodes. $G(V, E)$ is a 4-core graph where the vertices in the 5-cliques are far from Q. In contrast, the subgraph induced by thick solid edges are more relevant to Q. Even though it is a 3-core subgraph, the vertices in it are much closer to Q compared to $G(V, E)$. In order to identify the subgraphs with large k and small query distance like the 3-core subgraph, we define the kd-metric.

Definition 2.2. (KD-METRIC). Given a connected k-core G_k where $Q \subseteq V(G_k)$, the kd-metric of G_k is $\frac{k}{d}$ where $d = dist(G_k, Q)$.

The kd-metric identifies the subgraphs with large k and small query distance. For instance, the subgraphs induced by thick solid edges in Fig.2 are $\frac{k}{d}$ maximized.

Next, we improve the importance of Q. We define a proximity measure under which vertices close to Q have higher relevance. Then the vertices with lower relevance are removed.

Definition 2.3. (QUERY-BIASED RELEVANCE)

$$r(u) = \begin{cases} 1, & if\ u \in Q; \\ \alpha \cdot \sum_{v \in N(u)} \frac{r(v)}{|N(u)|}, & if\ u \in V \setminus Q; \end{cases}$$

In the definition, α $(0 < \alpha < 1)$ is the decay factor. Query-biased relevance (QBR) is a variant of penalized hitting probability (PHP) [30]. We use $N(u)$ instead of d_{max} (the maximum degree of a graph) as the divisor. Such a variant gives more relevance to the nodes close to Q especially when d_{max} is large and nodes in Q have low degree. Intuitively, QBR measures the average relevance coming from each edge. Fig.3 shows the relevance distribution under PageRank, PHP and QBR. Obviously, QBR shifts more relevance to the nodes around Q. As with PHP, $r(u)$ is no more than 1 and can be solved by the iterative method. In the rest of the paper, we denote $r_{G'}(v)$ as the QBR of v in G'.

Then we define our model for query-centralized community search as follows.

Definition 2.4. (KDR COMMUNITY). Given a graph $G(V, E)$ and a set of query nodes $Q \subseteq V(G)$, G' is a KDR community satisfying the following conditions:
(1) G' is a connected k-core with $Q \subseteq V(G')$.
(2) $\frac{k}{d}$ is maximized.
(3) $\frac{\sum_{v \in G'} deg(v) r_{G'}(v)}{|V(G')|(|V(G')|-1)}$ is maximized.

Conditions (1) and (2) provide a candidate subgraph for further refining and condition (3) extracts a densely connected query-centralized community from the subgraph. Notably, condition (3) maximizes the edge density $(\frac{2|E(G')|}{|V(G')|(|V(G')|-1)})$ if $r_{G'}(v)$ is removed.

The KDR model avoids the free rider effect. The free rider effect [17, 30] describes a community containing irrelevant subgraphs. Given a goodness metric $f(.)$ (the larger the better), assume an optimal solution H^* irrelevant to Q exists and a solution H to a community definition based on $f(.)$ is found; if $f(H \cup H^*) \geq f(H)$, then the definition suffers from the free rider effect. In the KDR model, there does not exist an optimal subgraph irrelevant to Q. If Q is not contained in a subgraph H, then $dist(H, Q) = +\infty$ and $\frac{k}{d} = 0$, which means the optimal subgraphs are all around Q.

Problem 1. (KD-PROBLEM). Given a graph $G(V, E)$ and a set of query nodes $Q \subseteq V(G)$, find a subgraph satisfying conditions (1) and (2) of Definition 2.4.

The KD-problem aims to find a candidate subgraph for the KDR-community. We can compute the optimal solution for it in $O(|E(G)^{2.5}|)$ (Section 3.1).

Problem 2. (KDR-PROBLEM). Given a graph $G(V, E)$ and a set of query nodes $Q \subseteq V(G)$, find a subgraph satisfying all the conditions of Definition 2.4.

THEOREM 2.5. *The KDR-problem is NP-hard.*

PROOF. Assume we maximize $\frac{2|E(G')|}{|V(G)|(|V(G')|-1)}$ based on the KD-problem. When $\frac{2|E(G')|}{|V(G)|(|V(G')|-1)} = 1$, the problem can be reduced from the Maximum Clique Decision problem which is NP-hard. The KDR-problem is even harder as the query-biased relevance is considered in the edge density and varies in different subgraphs. □

Algorithm 1: KD algorithm

```
   Input: Graph $G(V,E)$, query nodes $Q$
   Output: A subgraph $R$ for KD-problem
1  Find a maximal connected $k$-core subgraph $G_{k_{max}}$ containing
   $Q$ with largest $k$ and let $k_{max} \leftarrow k$, $l \leftarrow k_{max} - 1$;
2  $d_{min} = max\{dist_G(q, Q \setminus q) | q \in Q\}$;
3  Obtain $R_{k_{max}}$ from $G_{k_{max}}$; // Algorithm 2
4  $R \leftarrow R_{k_{max}}$, $kd_{temp} \leftarrow f(R_{k_{max}})$;
5  while $l \geq 1$ do
6      if $\frac{l}{d_{min}} \leq kd_{temp}$ then break;
7      Obtain $G_l$ by adding vertices of $cnum \geq l$ into $G_{l+1}$;
8      Obtain $R_l$ from $G_l$; // Algorithm 2
9      if $f(R_l) > kd_{temp}$ then $kd_{temp} \leftarrow f(R_l)$, $R \leftarrow R_l$;
10     $l \leftarrow l - 1$;
```

3 ALGORITHMS

In this section, we present an algorithm for the KD-problem. Then a locally exploring method for the KDR-problem is developed to fit in large networks. At last, we propose two strategies to accelerate the local method further.

3.1 KD Algorithm

Offline Operation. Before answering queries, we conduct k-core decomposition [3] on $G(V, E)$. $\forall v \in V(G)$, we record its core number $cnum$ $(max_{G' \subseteq G, v \in V(G')} \delta(G'))$ which is useful to build a connected k-core in online operation.
Online Operation. Algorithm 1 shows the procedure. Line 1 adds vertices of $cnum \geq t$ into Q, where $t = min\{cnum(q) | q \in Q\}$; if Q is not connected in the induced subgraph G_t, let $t \leftarrow t - 1$ and add vertices of $cnum \geq t$ iteratively until Q is connected in G_t. Then the lower bound of $dist(R, Q)$ is computed for pruning (Line 2). Through Algorithm 2 we obtain a k-core subgraph R_k of G_k with smallest $dist(R_k, Q)$ (Line 3). Thereafter, we enumerate R_l with smallest $dist(R_l, Q)$ and return the subgraph R of largest $\frac{k}{d}$ (Lines 5-10). Line 6 is a pruning condition. l is a decreasing number and $\forall G' \subseteq G, d_{min} \leq dist(G', Q)$, therefore, $f(R_l) \leq \frac{l}{d_{min}}$. A solution is found if the condition is satisfied. Algorithm 2 finds a connected k-core R_k containing Q with smallest $dist(R_k, Q)$. It is a k-core version of the Basic in [17].
Example. Given graph $G(V, E)$ and $Q = \{q_1, q_2, q_3\}$ in Fig.2(a), we firstly find the maximal 3-core subgraph G_3 $(G[V(G \setminus t)])$ and $d_{min} = 2$. Through Algorithm 2, $R_3 = G_3[G_3 \setminus \{p_1, p_2, p_3\}]$ is found with $f(R_3) = 1$. In the iteration, we find $l = 2$ and $\frac{l}{d_{min}} \leq f(R_3)$ which means a solution has been found. Therefore, we return R_3 as a solution.

THEOREM 3.1. *Algorithm 2 finds a connected k-core subgraph with the minimum query distance.*

PROOF. The proof is similar to the Basic algorithm of [17]. We omit it for brevity. □

THEOREM 3.2. *Algorithm 1 is exact for KD-problem.*

PROOF. In the worst case, the KD Algorithm finds the maximal connected t-core subgraph G_t containing Q (Line 7) and extracts R_t $(arg\,min_{R_t \subseteq G_t} dist(R_t, Q)$ and $\delta(R_t) \geq t)$

Algorithm 2: Reducing $dist(G_k, Q)$ of G_k

```
Input: Connected $k$-core $G_k$, query nodes $Q$
Output: $arg_{R_k \subseteq G_k, \delta(R_k) \geq \delta(G_k)} min\, dist(R_k, Q)$
1  $l \leftarrow 0$, $G_k^l \leftarrow G_k$;
2  while $Q$ is connected in $G_k^l$ do
3      $S \leftarrow \{v | arg\, max_{v \in G_k^l} dist_{G_k^l}(v, Q)\}$;
4      for $v \in S$ do
5          for $u \in N(v)$ do
6              $deg(u) \leftarrow deg(u) - 1$;
7              if $deg(u) < k$ then $S \leftarrow S \cup \{u\}$;
8      remove $S$ from $G_k^l$ and let $G_k^l$ be the connected
       component containing $Q$;
9      $G_k^{l+1} \leftarrow G_k^l$, $l \leftarrow l + 1$;
10 $R_k \leftarrow arg\, min_{G'_k \in \{G_k^0, \ldots, G_k^{l-1}\}} dist_{G'_k}(G'_k, Q)$;
```

from Algorithm 2 for each $t \in [1, k_{max}]$. We only need to show that at least one optimal solution exists in $\{R_t | t \in [1, k_{max}]\}$.

Assume no optimal solutions exist in $\{R_t | t \in [1, k_{max}]\}$, then a connected t-core subgraph G'_t ($G'_t \subset G_t$) exists and contains the optimal solution R'_t. By deleting vertices $G_t \setminus G'_t$ from G_t, we can obtain G'_t. Hence, $dist(R_t, Q) \leq dist(R'_t, Q)$ according to Theorem 3.1. If $\delta(R'_t) \leq \delta(R_t)$, then $f(R_t) \geq f(R'_t)$. Otherwise, R'_t can be obtained from G_l where $\delta(G_l) = \delta(R'_t)$. In this case, $\delta(R_l) \geq \delta(R'_t)$ and $dist(R_l, Q) \leq dist(R'_t, Q)$ (Theorem 3.1). Both the two cases contradict the assumption.

□

Theorem 3.3. *The KD algorithm runs in $O(|Q||E(G)|^{2.5})$.*

Proof. Let $m = |E(G)|$ and $n = |V(G)|$. Because of k-core property, the number of loops is no more than $\sqrt{m}$ ($k_{max} \leq \sqrt{m}$). In each loop, we delete one vertex in the worst case, so the loops $t \leq n$. In each loop, we should check the connectivity of Q ($O(m)$) and compute the query distance for each vertex ($O(|Q|m)$). Hence the loop takes $O(|Q|tm)$. As G is connected ($n = O(m)$), KD runs in $O(|Q|m^{2.5})$. □

3.2 Loc-kdr Algorithm

In order to solve KDR-problem, we develop a heuristic method called Loc-kdr to find a KDR-community locally. Algorithm 3 shows the main procedure of Loc-kdr.

Computing a Steiner tree. The Steiner tree is widely applied in community search for local exploration [2, 16, 17, 28]. It guarantees the connectivity of Q and can be a sketch of a community for expansion. 2-approximate algorithms [21, 25] are usually adopted for this NP-hard problem. However, applying them directly may produce a poor sketch. For instance, given graph $G(V, E)$ in Fig.2(a) and query $\{q_1, q_3\}$, the Steiner tree $T = \{(q_1, t), (q_3, t)\}$. T may produce a subgraph of small $\frac{k}{d}$. Thus we define the degree distance.

Definition 3.4. (DEGREE DISTANCE) . Given a graph $G(V, E)$ and an edge $(u, v) \in E(G)$, the degree distance of (u, v) is $dist(u, v) = \frac{2 \cdot \overline{degree}}{min\{deg(u), deg(v)\}} + 1$.

Algorithm 3: Loc-kd algorithm

```
Input: Graph $G(V, E)$, query nodes $Q$
Output: A subgraph $R$ of large $\frac{k}{d}$ and $\rho_w$
1 Compute a Steiner tree $T$ containing $Q$;
2 Obtain $G_{can}$ by expanding $T$ with vertices which are around
  $T$ and having small query distance;
3 Enlarge $\frac{k}{d}$ and $\rho_w$ of $G_{can}$;
4 Add vertices of large $\frac{k}{d}$ into $G_{can}$ and apply Line 3
  iteratively until large $\frac{k}{d}$ of $G_{can}$ and $\rho_w(G_{can})$ are reached;
```

Degree distance makes the Steiner tree tend to discover vertices in a denser subgraph. As vertices in the denser subgraph may be far from the query nodes, **1** is added to the distance and a numerator adjusts the two factors. The larger the numerator is, the more important density is in the distance function. Empirically, the numerator is twice the average degree of a graph. For instance, given $Q = \{q_1, q_3\}$ in Fig.2(a) ($2 \cdot \overline{degree}$=9.4), the total distance of $\{(q_1, t), (t, q_3)\}$ is 11.4 whereas the total distance of $\{(q_1, v_2), (v_2, v_5), (v_5, q_3)\}$ is 9.1. Obviously, the latter path is a better sketch.

Expanding the Steiner tree. We obtain a candidate subgraph by checking the neighborhood around T. As the subgraph induced from $V(T)$ may be a k-core with small k and query distance, we add vertices with small query distance to enlarge k. For each vertex v around T, if $dist_{T \cup \{v\}}(v, Q) \leq max_{u \in N(v) \cap T} dist_T(u, Q)$, we will add v to T. In the expanded graph G_{can}, $dist(G_{can}, Q) \leq dist(T, Q)$ is guaranteed.

Enlarging metrics $\frac{k}{d}$ and ρ_w. We improve the importance of Q during the procedure of enlarging $\frac{k}{d}$. We obtain G_k containing Q with the highest minimum degree. Then we conduct Algorithm 2 to find R_k. In each loop, we compute the query-biased relevance for each vertex in G_k^l, thereafter, we remove $min\{1, \epsilon \cdot |V(G_k^l)|\}$ vertices with smaller $deg(v) r_{G_k^l}(v)$ in S (Line 3). We choose $\epsilon = 0.03$ empirically.

Iteratively enlarging the metrics. We explore the vertices around G_{can} iteratively to enlarge the metrics. We collect vertices with $\frac{|N(v) \cap V(G_{can})|}{max\{dist(G_{can}, Q), dist_{v \cup G_{can}}(v, Q)\}} \geq f(G_{can})$. At the end of each iteration, we add the collected vertices into G_{can} and apply Step 3 to enlarge the metrics. The iteration is terminated once the new metrics are no longer enlarged.

Example. Given graph $G(V, E)$ in Fig.2(a) and $Q = \{q_1, q_3\}$, we get the Steiner tree T=$\{(q_1, v_2), (v_2, v_5), (v_5, q_3)\}$. Then we add S=$\{v_2, v_5, q_2, v_4, t\}$ into G_{can} in Step 2. In Step 3, we obtain $R_3 = G_{can}[V(G_{can}) \setminus t]$ with $\frac{k}{d} = 1$ and $\rho_w = 0.452$ if $\alpha = 0.9$. As none of the vertices surround R_3 have degree larger than 3, R_3 is returned.

Analyses. Query distance d is hard to reduce when k-core is maximized. Given G_{can} from Step 2, Step 3 also encounters the problem. Thus we use $max_{u \in N(v) \cap T} dist_T(u, Q)$ other than $dist(T, Q)$ in Step 2 to make the decrease of d easier. In Step 3, we iteratively remove the unqualified vertices to achieve larger metrics. As this step will be conducted many times, we can find large metrics in most cases. In order to control the size of community, we set an upper bound η for G_{can}. Let S denote the adding vertices, if $|S \cup G_{can}| > \eta$, only $\eta - |V(G_{can})|$ vertices with higher score in S are selected.

Algorithm 4: Pruning steiner tree computation

```
Input: Graph $G(V, E)$, query nodes $Q$
Output: A steiner tree $T$ containing $Q$
1  adjoin a vertex $q_0$ and edges $(q_0, q), q \in Q$ of length 0;
2  initialize $G'(V', E', D')$, where $V' = Q$, $E' = \phi$, $D' = \phi$;
3  $connected \leftarrow False$, $S \leftarrow \{q_0\}$, $PQ \leftarrow \phi$;
4  for $q \in Q$ do
5      $N(q) \leftarrow \{q\}$, $dist(s(q), q) \leftarrow 0$, $PQ.insert(q, s(q), 0)$,
       $maxDistance(q) \leftarrow \infty$;
6  while $PQ \neq \phi$ do
7      $(u, s(u), dist(u, s(u)) \leftarrow PQ.extractMinDistance()$;
8      for $v \in neighbour(u)$ do
9          if $v \notin S$ then
10             $dist_{max} \leftarrow getMaxDistance(s(v))$;
11             $dist(s(u), v) \leftarrow dist(s(u), u) + dist(u, v)$,
               $s(v) \leftarrow s(u)$;
12             if $dist(s(u), v) < dist_{max}$ then
13                 if $v \notin PQ$ then
14                     $PQ.insert(v, s(v), dist(s(v), v))$;
15                 else if $dist(s(u), v) <$ PQ.get(v).distance
                   then $PQ.update(v, s(v), dist(s(v), v))$;
16         else if $s(u) \neq s(v)$ then update edge $(s(u), s(v))$ in
           $G'(V', E', D')$ using Algorithm 5;
17     $S \leftarrow S \cup \{u\}$;
18 $T_1 \leftarrow MST(G')$ and build $G_2$ by replacing edges in $T_1$ with
   shortest paths in $G$;
19 $T_2 \leftarrow MST(G_2)$ and remove edges from $T_2$ to make the
   leaves are all in $Q$;
```

Thereafter, R is returned from Step 3 by operating the expanded G_{can}. The score is defined as $\frac{dist_{T\cup\{v\}}(v,Q)^2}{max_{u\in N(v)\cap T} dist_T(u,Q)}$ in Step 2 and $\frac{|N(v)\cap V(G_{can})|}{max\{dist(G_{can},Q), dist_{v\cup G_{can}}(v,Q)\}}$ in Step 4. The scores and strict rules of adding vertices in Step 2 and 4 enhance the edge density of G_{can}.

It is possible to use the KD Algorithm as a subroutine to get a KDR-community. However, the algorithm almost operates the whole graph when k is small which is obviously inefficient. Instead, we devise Loc-kdr to iteratively enlarge the metrics to approach the KDR-community.

Handling single-vertex-query. As one node may belong to communities of different densities [7], we handle single-vertex-query individually. Given one vertex q from G, we find all the connected subgraphs from the induced graph $G[N(q)]$. For each connected subgraph G_c, we conduct Step 2-4 of Algorithm 3 with $V(G_c) \cup q$.

3.3 Speed-up Strategies

Using Mehlhorn's method [25] with degree distance, Loc-kdr constructs a Steiner tree containing Q for expansion. However, as [25] visits the whole graph, Loc-kdr is still time-consuming for large graphs. The problem also exists in other local methods for community search [2, 16, 17].

Mehlhorn's method visits the whole graph in Step 1. It conducts a partition of $V(G)$ through a single shortest path computation. After the partition, $V = \cup_{q\in Q} n(q)$ and $\forall s, t \in Q, s \neq t, n(s) \cap n(t) = \emptyset$ where $n(q)$ is a set $S \subseteq V$ of

Algorithm 5: UpdateG'

```
Input: Vertex $u$, vertex $v$, $G'(V', E', D')$
Output: $G'(V', E', D')$
1  $dist(s(u), s(v))$=$dist(s(u), u)$+$dist(u, v)$+$dist(v, s(v))$;
2  if $(s(u), s(v)) \notin E'$ or $D_1(s(u), s(v)) > dist(s(u), s(v))$ then
3      update $(s(u), s(v), dist(s(u), s(v))$ in $G'$;
4      if $Q$ is connected in $G'$ and not connected then
5          $\forall q \in Q$, $maxDistance(q) \leftarrow dist_{G'}(q, Q \setminus q)$;
6          $connected \leftarrow True$;
```

vertices closer to $q \in Q$ and $s(v) = q$ if $v \in n(q)$. Thereafter, it constructs a complete distance graph $G_1 = (V_1, E_1, D_1)$, where $V_1 = Q$, $E_1 = \{(q_i, q_j)|q_i, q_j \in Q, qi \neq qj\}$, $D_1(q_i, q_j)$ is the minimum $dist(q_i, u) + dist(u, v) + dist(v, q_j)$ satisfying $u \in n(q_i)$ and $v \in n(q_j)$. The following steps are the same as Lines 18-19 of Algorithm 4.

In order to accelerate Step 1 of Mehlhorn's method, we propose two strategies in this section. The first one is a pruning strategy with provable guarantees and the second one is a heuristic strategy in a best-first-search manner.

3.3.1 *Pruning Strategy.*

Vertices from the same community are not far from each other in general. This strategy finds distance bound for each $q \in Q$ and once the exploring vertices exceed the bounds, exploration is terminated. Hence considerable computation can be avoided.

Algorithm 4 shows the procedure. Above all, q_0 is adjoined to each query node of length 0 for partitioning (Lines 6-17). Pruning is combined into the partition. Once a vertex u of the shortest distance is obtained (Line 7), $N(v)$ are considered (Lines 8-16). For each $v \in N(u)$, if v is not visited (Lines 9-15), a distance bound sup_d is found for v (initialized as $+\infty$, obtained from Algorithm 5), then $dist(s(v), v)$ is computed (Line 11). Triple $\{v, s(v), dist(v, s(v))\}$ will participate in further computation if $dist(v, s(v)) < sup_d$ otherwise be discarded. Algorithm 5 handles the condition that v is visited and $s(u) \neq s(v)$. If the condition in Line 2 is satisfied, triple $\{s(u), s(v), dist(s(u), s(v))\}$ will update $(s(u), s(v))$. The condition in Line 4 reveals that Q is connected in G', then sup_d $(dist_{G'}(q, Q \setminus q))$ of each $q \in Q$ is computed. After Lines 1-17 of Algorithm 4, the candidate edges of $MST(G_1)$ are all held. The next steps follow Steps 2-5 of [21].

Example. Consider $G(V, E)$ in Fig.2(a) and Q=$\{q_1, q_2\}$ (For clarity, **1** is the length for each edge). Above all, $n(q_1)$=$\{q_1\}$, $n(q_2)$=$\{q_2\}$ and triple $\{s(q_1), s(q_2), 1\}$ are found. Then the query nodes are connected and 1 is set to the distance bound (sup_d) for q_1 and q_2. $\forall v \in V \setminus \{q_1, q_2\}$, $dist(v, s(v)) \geq sup_d(s(v))$; graph G' with V'=$\{q_1, q_2\}$, E'=$\{(q_1, q_2)\}$ and D'=$\{dist(q_1, q_2)$=$1\}$ has all the edges in $MST(G_1)$.

As with [25], Algorithm 4 admits 2-approximation to the Steiner tree problem.

Theorem 3.5. *Given a connected graph $G' = (V', E', D')$, where $V' = Q$, the set of edges $E^* = \{(q_i, q_j)|(q_i, q_j) \in E', D'(q_i, q_j) > dist(q_i, Q \setminus q_i)\}$ are excluded from $MST(G')$.*

Proof. Assume there exists an edge $e_1 = (q_i, q_j) \in E^*$ in $G_1 = MST(G')$. Then $G_1 \setminus e_1$ is disconnected. Find a

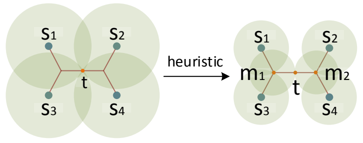


Figure 4: An instance of the heuristic strategy

shortest path P from q_i to q_j in G' and add each edge in P to $G_1 \setminus (q_i, q_j)$, denoted as G_2. As a result, G_2 is connected. Given $D'(e_1) > dist(q_i, Q \setminus q_i)$ and the total distance of P is no larger than $dist(q_i, Q \setminus q_i)$, the total distance of G_2 is less than that of $MST(G')$, a contradiction. □

THEOREM 3.6. *Given a connected graph $G' = (V', E', D')$, where $V' = Q$, consider an edge (q_i, q_j) is to be added into G', where the distance of (q_i, q_j) is equal to $dist(q_i, Q \setminus q_i)$. Then the newly added edge is unnecessary presented in $MST(G')$.*

PROOF. Similar to the proof of Theorem 3.5, there always exists a path P in G' from q_i to q_j to replace the newly added edge and the total distance of replaced spanning tree is no larger than the one with (q_i, q_j) included. □

COROLLARY 3.7. *Denote the distance graph computed from Algorithm 4 as G' and the one computed from Mehlhorn's Algorithm[25] as G_1, then $MST(G')$ is equivalent to $MST(G_1)$.*

PROOF. Algorithm 4 computes partition and triples simultaneously. Once Q in G' is connected, $\forall q \in Q$, $dist(q, Q \setminus q)$ is treated as the distance bound of q. Given Theorem 3.5 and 3.6, it is safe to prune vertex u having $dist(s(u), u) \geq dist(s(u), Q \setminus s(u))$. Consequently, the edges absent in G' are the ones that are not necessary in $MST(G_1)$. □

We also change the degree distance for the pruning strategy. Let v_k denote the vertex to visit, v_{k-1} denote the predecessor of v_k and s denote the source vertex. $dist(s, v_k) = c \cdot dist(s, v_{k-1}) + dist(v_k, v_{k-1})$ where $c > 1$. The pruning strategy benefits from such modification. In general, vertices far from Q are unlikely to share the same community with Q. Hence the distance increases with hops exponentially and vertices far from Q can be pruned easier. Besides, the distance is still gentle in the dense subgraph around Q. In Algorithm 4, the distance of a path $(q_i, ..., k_i, k_j, ..., q_j)$ is computed as $dist(q_i, q_j) = dist(q_i, k_i) + dist(k_i, k_j) + dist(k_j, q_j)$. Consider the graph in Fig.2(a); the shortest path from q_1 to q_3 is still $\{(q_1, v_2), (v_2, v_5), (v_5, q_3)\}$.

3.3.2 Heuristic Strategy. In this subsection, we provide a "scent" to guide the exploration in Section 3.3.1. Fig.4 shows the main idea of our heuristic strategy. Given graph $G(V, E)$ and $Q = \{s_1, s_2, s_3, s_4\}$, we assume the Steiner tree T consists of red paths (For clarity, we only present the important vertices and each edge is of equal length). In order to find T, the pruning strategy (the left one) has to explore the green shadow. If we provide "scent" for each $q \in Q$ and the "scent" can be propagated during the exploration, the "scent" of vertices (m_1 and m_2) residing in the collision zone of shadows can be enhanced. Consider vertices of strong "scent" are given priority access; the shadow (visited vertices) will be smaller than the pruning strategy when T is found. Based on this idea, our strategy is developed.

Table 2: Network statistics(K=10^3 and M=10^6)

Network	Abbr.	Domain	$\|V\|$	$\|E\|$	Diameter
Amazon	AZ	Product	335K	926K	44
DBLP	DP	Collaboration	317K	1M	21
Youtube	YT	Social	1.1M	3M	20
LiveJournal	LJ	Social	4M	35M	17
Orkut	OR	Social	3.1M	117M	9

We define scent similar to Definition 2.3 with $\frac{r(v)}{|N(v)|}$ instead of $\frac{r(v)}{|N(u)|}$. The scent of vertices closer or having short paths to Q is strong and hence guidance to short paths is provided. Next, we make three modifications to Algorithm 4.

• One more priority queue is applied for scent. We sort the inserted vertices using $\frac{r(v)}{|N(v)|}$ in descending order. It measures the average scent from each edge.

• For each iteration in Algorithm 4, the vertex u of minimum $max\{rank(dist_{com}), rank(scent)\}$ is extracted. Then u pushes $\alpha \cdot \frac{r(u)}{|N(u)|}$ forward to its neighbours. Afterwards, the distance and scent of each neighbour are updated. Notably, if the minimum hyper rank hr is larger than a predefined threshold $rank\_limit$, the vertex of top scent ranking is extracted out and tagged with $rank\_limit$.

• Once Q is connected in G', $r = min_{v \in G'} r(v)$ and $R = max_{v \in G'} hr(v)$ are obtained. For the extracted vertex u in each iteration, if $hr(u) \geq R$ and $r(u) \leq r$, the loop is over.

Analyses. Intuitively, vertices in a community C containing Q should be closer and densely connected to Q. $\frac{r(v)}{|N(v)|}$ punishes the vertices of extremely high degree and activates the vertices denoting themselves to C. The hyper rank considers both relevance and distance which also follows the intuition. Even though the strategy is heuristic, it has a good approximation for Steiner tree (Section 4.2).

4 EXPERIMENTS

We evaluate our model and algorithms in this section. Firstly, the experimental setup is introduced. Secondly, we discuss the performance of our solutions for varying queries. Thirdly, we compare Loc-kdr with popular solutions for community search on real and synthetic networks.

4.1 Experimental Setup

Real Networks. Table 2 summaries the real networks used in the experiments. These networks are publicly available from Stanford Network Analysis Project(*snap.stanford.edu*). The networks provide ground-truth communities [31] and have been used for evaluation in prior works of community search [1, 7, 10, 15–17, 30].

Synthetic Networks. We also generate synthetic networks using the well-known benchmark LFR [22] for evaluation. The fixed parameters in the synthetic networks are the exponents of degree ($\gamma = 2$) and community size ($\beta = 1$) in the power law distribution, the range of the community size (from 10 to 100) and the average degree of vertices ($\overline{d} = 10$). We vary n (number of vertices), u (the proportion of neighbors around a vertex residing in other communities) and on (number of vertices belonging to two or more communities) for evaluation.

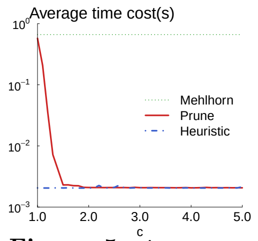


Figure 5: Average time of speed-up strategies

	$\frac{k}{d}$	ρ_w
Mehlhorn	0.935	0.465
Pruning	0.935	0.465
Heuristic	0.935	0.467

Table 3: Metrics of speed-up strategies

Evaluation Criteria. We focus on t (running time), F_1, $\frac{k}{d}$ and ρ_w in our experiments. $F_1 = \frac{2\cdot|C \cap C_T|}{|C|+|C_T|}$ measures the similarity between a discovered community C and a ground-truth community C_T. kd-metric $\frac{k}{d}$ and weighted edge density ρ_w are the metrics of KDR model (Definition 2.4). We use them to measure the quality of query-centralized communities.
Baselines. We select MDC [28], GrCon [2] and LCTC [17] for comparison. They are the most related works to our query-centralized version of community search. Notably, LCTC achieves the best F_1 compared to QDC [30] and methods for local community detection in the real networks [17, 30]. Similar to PHP [30], we set $\alpha = 0.9$ for query-biased relevance. Besides, we set $\eta = 200$ for the size bound and $rank\_limit = 50$ for the heuristic strategy.
Hardware. Algorithms[1] are written in C++ based on igraph [6] and run in main memory. Besides, experiments are conducted on a Linux Server with 128GB main memory and Intel Xeon CPU E5-2630 (2.4GHz).

4.2 Performance for Different Queries

We test our approaches using different queries on DBLP. We vary query size $|Q|$, degree rank Q_d and inter-distance l to generate different sets of queries. $|Q|$ represents the number of query nodes in a query. A node with $Q_d = X\%$ means its degree is larger than $(100 - X)\%$ vertices in the network. We use Q_d to vary the degrees of Q. l is the max shortest distance between any two query nodes. By default, we set $|Q| = 3$, $Q_d = 80\%$ and $l = 2$.
Speed-up strategies evaluation. We randomly select 100 queries using the default settings. Then we compare the performance of Loc-kdr based on different methods connecting query nodes (Mehlhorn [25], pruning and heuristic strategies). We vary c of the modified distance in Section 3.3.1 to evaluate the speed-up strategies. Loc-kdr using Mehlhorn's method is also tested using the degree distance (Definition 3.4) as a reference. Table 3 shows the metrics. As Loc-kdr based on the pruning strategy obtains almost the same metrics with different c, we only present one line corresponded with "Pruning" in Table 3. Similarly, we only present one line corresponded with "Heuristic" in Table 3. Fig.5 reports the average time. Both Pruning and Heuristic are fast due to their local property. As we analyze in Section 3.3.1, the speed of Pruning is sensitive to c. The heuristic is stable with different c. In addition, we also test Heuristic with $\alpha \in [0.5, 1.0]$

[1]The code is available at https://github.com/ZhuoWang2018/Query-centralized-Community-Search

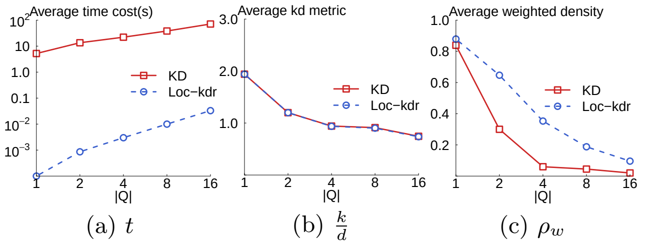


Figure 6: Varying $|Q|$

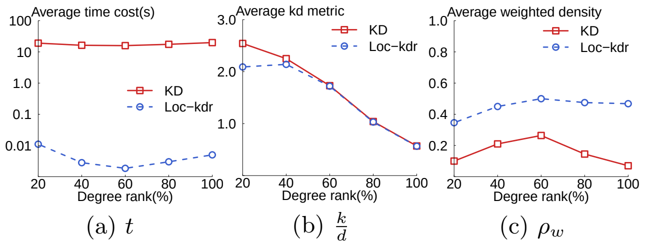


Figure 7: Varying degree rank

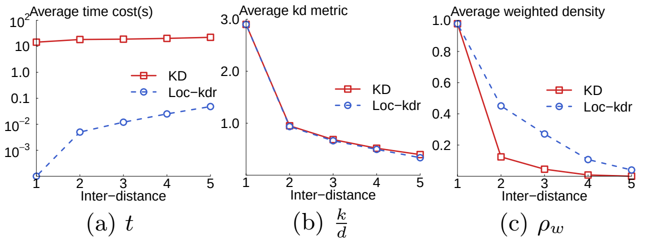


Figure 8: Varying inter-distance

and find the same results with Table 3. Hence the heuristic is robust and has a good approximation to Mehlhorn's method and the pruning strategy. In the following experiments, the heuristic strategy is applied in Loc-kdr by default.
Varying query size. We test $|Q|$ in $\{1, 2, 4, 8, 16\}$. For each $|Q|$, we randomly select 100 queries and report the results in Fig.6. As Loc-kdr produces multiple communities of single query node, we use the one of max $\frac{k}{d}$ for evaluation. Loc-kdr is much faster than KD. In addition to good approximation of $\frac{k}{d}$, Loc-kdr achieves better ρ_w than KD.
Varying degree rank. We sort the vertices by the degree in descending order and partition them into five equal-sized buckets. For each bucket, we randomly select 100 queries. Fig.7 shows the results. Loc-kdr is faster. When Q_d (degree rank) is small, Loc-kdr needs more iterations to enlarge metrics. Otherwise, it spends more time in connecting queries. Therefore, Loc-kdr is faster when $Q_d = 60\%$. Due to the strict expansion of Loc-kdr, it has higher ρ_w but gets a slight decrease in $\frac{k}{d}$ when Q_d is small.
Varying inter-distance. We vary l from 1 to 5. For each l, we select 100 queries randomly. Fig.8 shows the performance of KD and Loc-kdr. Loc-kdr is faster than KD. It is robust and of better ρ_w than KD when l is varied.

4.3 Comparison in Real Networks

We compare Loc-kdr with MDC, GrCon and LCTC in real networks (Table 2). For each network, we randomly select 1,000 samples from the ground-truth communities. Then we

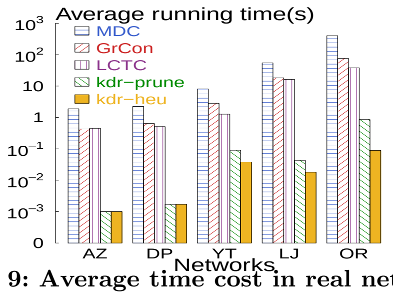


Figure 9: Average time cost in real networks

Table 4: Average metrics in real networks

	Average F_1 (%)				Average ρ_w			
	MDC	GrCon	LCTC	Loc-kdr	MDC	GrCon	LCTC	Loc-kdr
AZ	59.6	83.2	**91.6**	91.1	0.241	0.581	0.671	**0.677**
DP	44.5	82	86.2	**90.2**	0.183	0.605	0.658	**0.679**
YT	30.7	48.4	61.8	**68.3**	0.183	0.289	0.532	**0.561**
LJ	62.1	68	79	**81.6**	0.381	0.551	0.648	**0.667**
OR	40.4	47.6	39.4	**53.5**	0.146	0.15	0.168	**0.195**

randomly select $|Q| \in [1, 16]$ nodes as a query for each sample. We evaluate average time, F_1 and ρ_w over all the queries.

Fig.9 reports the average running time. We present the speed of Loc-kdr with two speed-up strategies. Loc-kdr runs much faster than the other methods. Besides, heuristic-based Loc-kdr is faster than pruning-based Loc-kdr in large real networks (we set $c = 3.0$ for the pruning strategy).

Table 4 shows the average F_1 and ρ_w. Loc-kdr produces multiple communities for single query node. For fairness, the metrics are collected for $|Q| \in [2, 16]$. The number of such queries is about 850 for each network. Pruning and heuristic based Loc-kdrs achieve similar results and hence we only show the results for Loc-kdr based on heuristic strategy.

In all cases, Loc-kdr outperforms the other methods in terms of ρ_w. The other k-core/truss based methods cannot guarantee ρ_w when the query nodes are of low degree. LCTC is the second best method. Due to the k-truss property, it discovers denser subgraphs than GrCon and MDC.

Loc-kdr achieves best F_1 in most cases which reveals that Loc-kdr discovers query-dependent communities effectively. Besides, the improvement of F_1 shows that considerable communities are not density maximized. We also find LCTC achieves slightly better F_1 than Loc-kdr in Amazon. Different from the other social networks, Amazon is a hierarchical co-purchasing network of large diameter. Some ground-truth communities contain members less relevant to query nodes but of small query distance. Loc-kdr removes the members due to the demand for query centralization. However, the members are hard to remove in LCTC and thus preserved.

4.4 Comparison in Synthetic Networks

We also evaluate the methods in synthetic networks generated by the LFR benchmark [22]. Based on the default settings (Section 4.1), we generate two sets of networks for evaluation. Query generation and evaluation are the same as the ones in Section 4.3 (The number of queries with $|Q| > 1$ is about 950 for each network). Fig.10 and Fig.11 report the results.

Both u and on affect the clearness of the community structure. As u increases, F_1 decreases for each method. Besides,

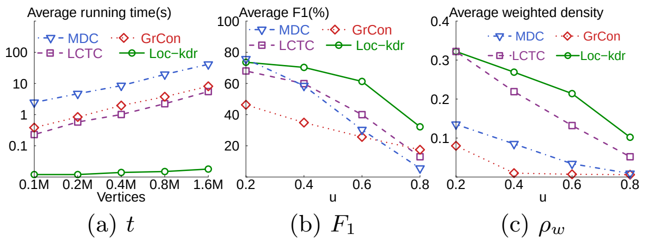


Figure 10: Evaluation on the networks with $on = 0.01n$. The networks in (a) are generated with fixed $u = 0.4$ and vertices n varying from $0.1M$ to $1.6M$. Networks in (b) and (c) are generated with fixed vertices $n = 100K$ and u ranging from 0.2 to 0.8.

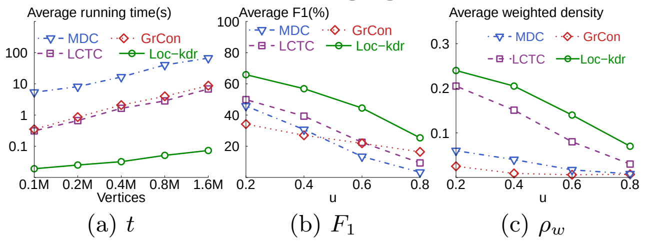


Figure 11: Evaluation on the networks with $on = 0.1n$. The other parameters are the same with Fig.10.

F_1 in Fig.11 is smaller than that in Fig.10. Due to the maintenance of graph structures, ρ_w also decreases with increasing u and larger on. Compared to other methods, Loc-kdr is more robust and achieves the best performance. This confirms the effect of our query-centralized model and approaches.

Fig.10(a) and Fig.11(a) show the efficiency of the selected methods. The average time increases with n. Loc-kdr is much faster than the other methods. Because of locally connecting query nodes and discovering communities, the time of Loc-kdr increases slowly. This confirms the scalability of Loc-kdr.

5 RELATED WORK

Community search [28] studies local community detection [5, 20] from the perspective of combinatorial optimization. Recently various community models have been studied based on dense graph structures such as k-core [2, 8, 10, 11, 28], quasi-clique [7], k-truss [1, 15–17] and query-biased densest subgraph [30]. Among them, studies [2, 17, 28, 30] handling multiple query nodes in simple graphs are the most related works to ours. Minimum degree community (MDC) [28] discovers a k-core subgraph with largest k containing query vertices globally under the distance and size constraints. Barbieri et al. [2] minimize the size of the subgraph based on MDC and devises indices for their local method GrCon (Greedy Connection). Query-biased densest community (QDC) [30] shifts density around the query nodes by developing penalized hitting probability. Huang et al. [17] defines k-truss communities of largest k and smallest diameter (Closest Truss Community). Then the method LCTC is developed for CTC. Distinguished from these works, our model (KDR) is designed for the query-centralized community where the query nodes play important roles in the community. For one thing, KDR-community is a connected k-core maximizing $\frac{k}{d}$ which can find dense communities with small query distance while avoiding the free rider effect. For another, the weighted edge density can discover members largely influencing/influenced by the query nodes, which ensures the centrality of the query.

In community search, the Steiner tree can be used as a sketch for local exploration while ensuring the connectivity of the query nodes [2, 16, 17, 28, 30]. 2-approximate methods [21, 25] are widely applied to the NP-hard problem. However, the methods still visit the whole graph which is time-consuming. Considering the feature that query nodes sharing a common community are close to each other, we develop two speed-up strategies. Compared to the index-based [13, 14, 23] and breath-first [4, 18, 19] heuristics, our strategies can visit tiny amount of vertices to connect query nodes with no indices.

There exist several studies finding interesting subgraphs given a set of query nodes. Studies [9, 26, 29] find a subgraph connecting the query nodes. [12] discovers vertices largely influencing the query nodes. Different from these works, our query-centralized community search not only provides good connections among the query nodes but also discovers a personalized community including the members largely influencing or influenced by the query nodes.

Recently, attributed [11, 16], spatial [10] and influential [24] community search have been studied. We will extend the KDR model to these directions in the future work.

6 CONCLUSION

In this paper, we study the query-centralized community search problem. Given a set of query nodes Q, we find a densely connected community in which the query nodes play important roles. We formulate the community as a connected k-core subgraph G_s maximizing $\frac{k}{d}$ where d is the query distance, besides, G_s has the largest query-biased edge density. Then we develop methods (KD and Loc-kdr) to solve the related problems. We also propose two speed-up strategies to make Loc-kdr scalable to large networks. Experimental results on large real and synthetic networks demonstrate the performance of our solutions for the query-centralized community search.

REFERENCES

[1] Esra Akbas and Peixiang Zhao. 2017. Truss-based Community Search: a Truss-equivalence Based Indexing Approach. In *International Conference on Very Large Data Bases, VLDB*.

[2] Nicola Barbieri, Francesco Bonchi, Edoardo Galimberti, and Francesco Gullo. 2015. Efficient and effective community search. *Data Mining and Knowledge Discovery* 29, 5 (2015), 1–28.

[3] V. Batagelj and M. Zaversnik. 2003. An O(m) Algorithm for Cores Decomposition of Networks. *Computer Science* 1, 6 (2003), 34–37.

[4] Gaurav Bhalotia, Arvind Hulgeri, Charuta Nakhe, Soumen Chakrabarti, and S Sudarshan. 2002. Keyword searching and browsing in databases using BANKS. (2002), 431–440.

[5] Aaron Clauset. 2005. Finding local community structure in networks. *Physical Review E* 72, 2 (2005), 026132.

[6] Gabor Csardi and Tamas Nepusz. 2006. The Igraph Software Package for Complex Network Research. *Inter J Complex Sys* complex systems (2006).

[7] Wanyun Cui, Yanghua Xiao, Haixun Wang, Yiqi Lu, and Wei Wang. 2013. Online search of overlapping communities. In *ACM SIGMOD International Conference on Management of Data*. 277–288.

[8] Wanyun Cui, Yanghua Xiao, Haixun Wang, and Wei Wang. 2014. Local search of communities in large graphs. In *ACM SIGMOD International Conference on Management of Data*. 991–1002.

[9] Christos Faloutsos, Kevin S Mccurley, and Andrew Tomkins. 2004. Fast discovery of connection subgraphs. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2004), 118–127.

[10] Yixiang Fang, Reynold Cheng, Xiaodong Li, Siqiang Luo, and Jiafeng Hu. 2017. Effective Community Search over Large Spatial Graphs. *Proceedings of the Vldb Endowment* 10, 6 (2017), 709–720.

[11] Yixiang Fang, Reynold Cheng, Siqiang Luo, and Jiafeng Hu. 2016. Effective community search for large attributed graphs. *Proceedings of the Vldb Endowment* 9, 12 (2016), 1233–1244.

[12] Aristides Gionis, Michael Mathioudakis, and Antti Ukkonen. 2015. Bump hunting in the dark: Local discrepancy maximization on graphs. In *IEEE International Conference on Data Engineering*. 1155–1166.

[13] Andrey Gubichev and Thomas Neumann. 2012. Fast approximation of steiner trees in large graphs. (2012), 1497–1501.

[14] Hao He, Haixun Wang, Jun Yang, and Philip S Yu. 2007. BLINKS: ranked keyword searches on graphs. (2007), 305–316.

[15] Xin Huang, Hong Cheng, Lu Qin, Wentao Tian, and Jeffrey Xu Yu. 2014. Querying k-truss community in large and dynamic graphs. In *ACM SIGMOD International Conference on Management of Data*. 1311–1322.

[16] Xin Huang and Laks V. S Lakshmanan. 2017. Attribute-driven community search. *Proceedings of the Vldb Endowment* 10 (2017).

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

[18] Varun Kacholia, Shashank Pandit, Soumen Chakrabarti, S Sudarshan, Rushi Desai, and Hrishikesh Karambelkar. 2005. Bidirectional expansion for keyword search on graph databases. *very large data bases* (2005), 505–516.

[19] Gjergji Kasneci, Maya Ramanath, Mauro Sozio, Fabian M Suchanek, and Gerhard Weikum. 2009. STAR: Steiner-Tree Approximation in Relationship Graphs. (2009), 868–879.

[20] Isabel M Kloumann and Jon M Kleinberg. 2014. Community membership identification from small seed sets. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1366–1375.

[21] L. Kou, G. Markowsky, and L. Berman. 1981. A fast algorithm for Steiner trees. *Acta Informatica* 15, 2 (1981), 141–145.

[22] Andrea Lancichinetti, Santo Fortunato, and Filippo Radicchi. 2008. Benchmark graphs for testing community detection algorithms. *Physical Review E* 78, 4 (2008), 046110.

[23] Guoliang Li, Beng Chin Ooi, Jianhua Feng, Jianyong Wang, and Lizhu Zhou. 2008. EASE: an effective 3-in-1 keyword search method for unstructured, semi-structured and structured data. (2008), 903–914.

[24] Rong Hua Li, Lu Qin, Jeffrey Xu Yu, and Rui Mao. 2015. Influential community search in large networks. *Proceedings of the Vldb Endowment* 8, 5 (2015), 509–520.

[25] Kurt Mehlhorn. 1988. A faster approximation algorithm for the steiner problem in graphs. *Inform. Process. Lett.* 27, 3 (1988), 125–128.

[26] Natali Ruchansky, Francesco Bonchi, David Garciasoriano, Francesco Gullo, and Nicolas Kourtellis. 2015. The Minimum Wiener Connector Problem. *international conference on management of data* (2015), 1587–1602.

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

[28] Mauro Sozio and Aristides Gionis. 2010. The community-search problem and how to plan a successful cocktail party. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 939–948.

[29] Hanghang Tong and Christos Faloutsos. 2006. Center-piece subgraphs: problem definition and fast solutions. *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (2006), 404–413.

[30] Yubao Wu, Ruoming Jin, Jing Li, and Xiang Zhang. 2015. Robust local community detection: on free rider effect and its elimination. *Proceedings of the Vldb Endowment* 8, 7 (2015), 798–809.

[31] Jaewon Yang and Jure Leskovec. 2015. Defining and evaluating network communities based on ground-truth. *Knowledge and Information Systems* 42, 1 (2015), 181–213.