Efficient Radius-bounded Community Search in Geo-social Networks

Journal:	*Transactions on Knowledge and Data Engineering*
Manuscript ID	TKDE-2020-10-1186
Manuscript Type:	Regular
Keywords:	K-core, Geo-social Network, Community search, Diversification

Efficient Radius-bounded Community Search in Geo-social Networks

Kai Wang, Shuting Wang, Xin Cao, Wenjie Zhang, and Lu Qin

Abstract—Driven by real-life applications in geo-social networks, we study the problem of computing radius-bounded k-cores (RB-k-cores) that aims to find communities satisfying both social and spatial constraints. In particular, the model k-core (i.e., the subgraph where each vertex has at least k neighbors) is used to ensure the social cohesiveness, and a radius-bounded circle is used to restrict the locations of users in an RB-k-core. We explore several algorithmic paradigms to compute RB-k-cores, including a triple-vertex-based paradigm, a binary-vertex-based paradigm, and a paradigm utilizing the concept of rotating circles. The rotating-circle-based paradigm is further enhanced by several pruning techniques to achieve better efficiency. In addition, to find representative RB-k-cores, we study the diversified radius-bounded k-core search problem, which finds t RB-k-cores to cover the most number of vertices. We first propose a baseline algorithm that identifies the distinctive RB-k-cores after finding all the RB-k-cores. Beyond this, we design algorithms that can efficiently maintain the top-t candidate RB-k-cores and also achieve a guaranteed approximation ratio. Experimental studies on both real and synthetic datasets demonstrate that our proposed techniques can efficiently compute (diversified) RB-k-cores. Moreover, our techniques can be used to compute the minimum-circle-bounded k-core and significantly outperform the existing techniques.

Index Terms—K-core, Geo-social network, Community search, Diversification

1 INTRODUCTION

With the wide availability of wireless communication techniques and GPS-equipped mobile devices (e.g., smartphones and tablets), people can now easily access the internet. This leads to the emergence of geo-social networks, such as Twitter and Foursquare, where social networks are combined with users' geo-spatial information. Consequently, retrieving subgraphs with high cohesiveness in geo-spatial social networks has become a popular research topic recently [14], [42], [44].

In this paper, we study the problem of efficiently computing *radius-bounded cohesive subgraphs* in a geo-spatial social network G (or abbreviated as a geo-social network). That is, given a vertex q in G and a radius r, find all cohesive subgraphs g of G such that g contains q and all vertices in g fall into a circle with the radius r. There are many types of cohesive subgraph models in the literature such as k-core [32], k-truss [8] and clique [25]. While our proposed framework generally works for various cohesive subgraphs, in this paper, we present our work restricted to a specific cohesive subgraph k-core [32] where each vertex has at least k neighbors.

Applications. The problem of computing *radius-bounded k-cores*, namely RB-k-cores, has many real real-life applications. On social platforms like Facebook and Twitter, personalized event recommendation is an essential part. For example, "Events for you" is a valuable Facebook component that recommends events to users based on their locations and social connection. Nevertheless, the current technology cannot provide a service of Events-For-You based on users' arbitrary requests. For example, the Events-For-You based on Leo's following request cannot be accommodated by the current technology. In this example, Leo wants to hold a party (an Events-For-You activity) to play board games (e.g., Monopoly, Uno, and Risk) by gathering a group of people who are not living far away (say, bounded by a circle with a radius r) and each of whom has many friends in the group (say, at least k friends). Figure 1 shows a geo-social network where vertices represent users, edges represent friendships, and locations represent the home locations of users. If we set $r = 3$ and $k = 3$, there are two RB-k-cores recommended to Leo as illustrated by the shadow area, i.e., {Leo, Ken, Jim, Adam} and {Leo, Bill, Frank, Bob, Lee}. This is a typical example of computing RB-k-cores. In addition, when users propose query requests, there may exist a large number of RB-k-cores that satisfy the query constraints. In these circumstances, the system only needs to recommend representative communities with rich information to users.

Moreover, as studied in [14], [27], people with close social relationships tend to purchase in places that are also close. For this application, the locations of users represent places in a geo-social network. To boost sales figures, advertisement messages can be sent to the RB-k-core of customers. For instance, if we want to promote an item A, the system can advertise A to the RB-k-core members (customers) using the query point (customers) who purchased A.

Existing Studies. Several studies over community retrieval exist in geo-social networks, but they are all different from our problem. In the literature, various models including k-core [32], k-truss [8], and clique [25] have been studied to

- *Kai Wang, Xin Cao, and Wenjie Zhang are with the University of New South Wales, Australia. E-mail: kai.wang@unsw.edu.au, xin.cao@unsw.edu.au, zhangw@cse.unsw.edu.au.*
- *Shuting Wang is with Zhejiang Gongshang University, China. Email: wangshuting@zjgsu.edu.cn.*
- *Lu Qin is with the University of Technology Sydney, Australia. Email: lu.qin@uts.edu.au.*

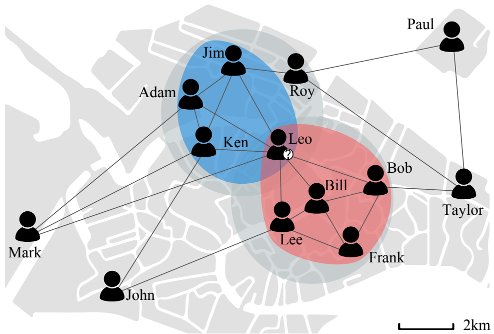


Figure 1: A geo-social network

retrieve cohesive subgraphs without considering the spatial information of users. Thus, these models are not applicable to compute RB-k-cores. For example, in Figure 1, Mark will be added into the community formed by {Leo, Ken, Adam, Jim} if we use the model k-core for $k = 3$ though Mark is far away from the other users. On the other hand, [16], [38] find a group of spatial objects without considering the network information, and thus they also cannot solve our problem RB-k-cores. In Figure 1, Roy will be added into the community formed by {Leo, Ken, Jim, Adam} using spatial information (bounded by a circle) only, but the network connections between Roy and the other people are fragile.

The most closely related works can be found in [14], [42], [44], which consider both social constraint (network structure) and spatial constraint in retrieving communities. In particular, they all use k-core to ensure the social (structure) cohesiveness of communities in a network. Zhang et al. [42] study the community detection problem, which uses pairwise similarity (distance) between each pair of vertices to ensure the spatial cohesiveness of communities while computing the maximum k-core or all maximal k-cores. As studied in [36], our problem is inherently different from the problem in [42]; that is, the generated results are very different. Moreover, our problem RB-k-cores is PTIME, while the problem in [42] is NP-hard. Zhu et al. in [44] study the problem of finding the maximum k-core in a given rectangle containing a query vertex. They also study the problem of finding the k-core with exact (or no less than) c vertices such that the longest distance from these vertices to q is minimized. These problems in [44] are also different from ours. Fang et al. in [14] study the problem of computing the k-core containing the query vertex covered by the smallest circle. While the problem in [14] is different from our problem RB-k-cores, as a byproduct, our techniques can be applied to the problem in [14] and can achieve a speed-up around twice.

Challenges. The main challenge of efficiently computing RB-k-cores is threefold.

1) The location of the radius-bounded circle of a RB-k-core is unknown. Therefore, it is a challenge to enumerate such circles efficiently.
2) In the process of finding all the RB-k-cores, it is cost-prohibitive to construct and verify the candidate subgraphs individually. Therefore, it is important to reuse intermediate computation results and explore possible cost-sharing, which is challenging.
3) Given a query request, there may exist many RB-k-cores in the result set. Thus, when users only want to retrieve the representative RB-k-cores rather than all of them, it is also challenging to identify such distinctive RB-k-cores with rich information efficiently.

Contributions. We first explore three paradigms to retrieve all the RB-k-cores in this paper. The first paradigm is triple-vertex-based algorithm (TriV) inspired by [14]. It proposes to firstly generate all candidate circles containing q, secondly check the corresponding radius to verify the given radius bound, and then compute the maximum k-core for the vertices in each candidate circle. To avoid generating too many candidate circles or missing results, TriV generates all the candidate circles by enumerating all the triple-vertex- and binary-vertex-combinations.

To reduce the number of candidate circles, we further propose the binary-vertex-based algorithm (BinV). In BinV, we effectively use the given parameter r and only generate the circles with a radius r such that the circle arc passes a pair of vertices in G (for every pair). Generally, for each pair of vertices, we have at most two such circles. This guarantees to generate $O(n^2)$ candidate circles and reduces $O(n^3)$ candidate circles in TriV to $O(n^2)$.

We can observe that there are many reusable intermediate computation results in the process of finding RB-k-cores. The third paradigm is to share computation costs among the computation of RB-k-cores. To effectively share the computation, we design the rotating-circle-based algorithm (RotC) so that the computation in BinV can be shared among the "adjacent" circles. Specifically, we fix a vertex u for each vertex in G then check the remaining vertices v such that for each pair u and v, we use BinV to generate the two circles with radius r (maybe degenerate to one if r is half of the distance between u and v). Then, we will share the computation among the adjacent circles.

To find distinctive RB-k-cores, we study the diversified RB-k-core search problem to find t representative RB-k-cores that cover the most number of vertices. This problem can be recognized as the max k-cover problem [15], which is NP-hard. A simple greedy solution DivBS is that after obtaining all the RB-k-cores, we run t iterations and identify the RB-k-core, which covers the maximum number of uncovered vertices for each iteration. Although DivBS can have a good approximation ratio, the main drawback of DivBS is that it needs to compute all the RB-k-cores firstly, which isolate the computation of RB-k-cores and the finding of diversified top-t RB-k-cores. Towards this issue, we propose the DivRotC$^+$ algorithm that maintains the top-t candidates in the RB-k-core computing process and achieves a guaranteed approximation ratio. Also, several useful pruning techniques are deployed into DivRotC$^+$ to enhance the performance further.

Our principal contributions are summarized as follows.

- We propose the RB-k-core model and develop a novel paradigm to compute RB-k-cores to share computations among different RB-k-cores.
- We propose several new optimization techniques that speed up the computation of finding all the RB-k-cores.
- We propose efficient algorithms along with several dedicated pruning techniques to find distinctive RB-k-cores with rich information.

- Extending our algorithms to the problem in [14] can achieve a speed-up around twice.
- We conduct comprehensive experiments on real geo-social networks to evaluate our algorithms.

Organization. The rest of the paper is organized as follows. Section 2 presents the preliminaries. Section 3 introduces our techniques to solve the RB-k-core search problem. The study of the diversified RB-k-core search problem is presented in Section 4. Section 5 reports experimental results. Section 6 reviews the related work. Section 7 concludes the paper.

2 PROBLEM DEFINITION

In this section, we formally introduce fundamental concepts and definitions. Mathematical notations used throughout this paper are summarized in Table 1.

Table 1: The summary of notations

Notation	Definition
G	a geo-social graph
G_k^r, B	a RB-k-core
u, v	vertices in the geo-social graph
$deg_G(v)$	the degree of vertex v in G
$N_G(v)$	the set of neighbors of vertex v in G
$d(u, v)$	the Euclidean distance between u and v
$O(c, \gamma)$	a circle centered at c with radius γ
$g(c, \alpha)$	a square centered at c with side length α
X, S	a set of vertices
$G(S)$	an induced subgraph of S
$\mathcal{R}$	the result RB-k-core set
$\mathcal{D}$	a set of diversified RB-k-cores

Our problem is defined over a geo-social graph $G(V, E)$, where $V(G)$ denotes the vertex set, and $E(G)$ denotes the edge set. The vertices represent the social network users and the edges represent their relationships in geo-social networks. Each vertex $v \in V(G)$ has a location $(v.x, v.y)$ which denotes the position of v along x- and y-axis in a two-dimensional space and the vertices are static in our problem. The Euclidean distance between u and v is denoted as $d(u, v)$. We denote the set of neighbors of each vertex v in G by $N_G(v) = \{u \in V(G) \mid (v, u) \in E(G)\}$ and the degree of vertex v by $deg_G(v) = |N_G(v)|$. We denote a circle centered at c with radius γ as $O(c, \gamma)$. Given a set of vertices $S \subseteq V(G)$, we use $G(S)$ to denote an induced subgraph of G formed from S such that $G(S) = (S, \{(u, v) \in E(G) \mid u, v \in S\})$.

Before formally defining the problem, we first introduce the following critical concepts to describe the social constraint and the spatial constraint.

Definition 1 (k-Core). Given a graph G and a positive integer k, the k-core of G denoted as H_k is the maximal subgraph of G, where $deg_{H_k}(v) \geq k$, for each $v \in V(H_k)$.

Based on the k-core concept, we ensure the social constraint by restricting the minimal degree of vertices in a RB-k-core. Note that our proposed solutions can be easily adapted to other cohesive structure concepts (e.g., k-truss [8], clique [25]), which can be used to define the social constraint from different perspectives.

Definition 2 (Minimum Covering Circle (MCC)). Given a set of vertices S, the minimum covering circle of S is the circle, which encloses all the vertices $v \in S$ with the smallest radius. We call the vertices which lie on the boundary of an MCC the boundary vertices.

After introducing k-core and MCC, we are ready to define the RB-k-core as follows.

Definition 3 (Radius-Bounded k-Core). Given a geo-social graph $G(V, E)$, a vertex $q \in V(G)$, a positive integer k and a query radius r, a subgraph of G denoted by G_k^r is a Radius-Bounded k-Core, if it satisfies the following constraints:

1) **Connectivity constraint.** $G_k^r \subseteq G$ is connected and contains q;
2) **Social constraint.** $\forall v \in V(G_k^r)$, $deg_{G_k^r}(v) \geq k$;
3) **Spatial constraint.** The MCC of $V(G_k^r)$ has a radius $r' \leq r$;
4) **Maximality constraint.** There exists no other supergraph $G'^r_k \supset G_k^r$ satisfying (1), (2), and (3).

Problem Statement (RB-k-core Search) Given a geo-social graph $G(V, E)$, a vertex $q \in V(G)$, a positive integer k, and a query radius r, our *RB-k-core search problem* aims to return all the RB-k-cores in G.

We then define coverage as follows.

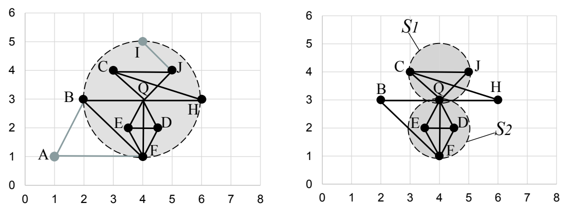
Definition 4 (Coverage). Given a set of RB-k-cores $\mathcal{D} = \{B_1, B_2, ...\}$ in G, the coverage of $\mathcal{D}$ denoted by $cov(\mathcal{D})$ is the set of vertices covered by the RB-k-cores in $\mathcal{D}$, i.e., $cov(\mathcal{D}) = \bigcup_{B \in \mathcal{D}} V(B)$.

Problem Statement (Diversified top-t RB-k-core Search) Given a geo-social graph $G(V, E)$, a vertex $q \in V(G)$, positive integers k and t, and a query radius r, our *diversified top-t RB-k-core search problem* (or short as diversified RB-k-core search problem) aims to return a set $\mathcal{D}$ of RB-k-cores in G, such that (1) each $B \in \mathcal{D}$ is a RB-k-core, (2) $|\mathcal{D}| \leq t$, and (3) $|cov(\mathcal{D})|$ is maximized. The set $\mathcal{D}$ contains the diversified top-t RB-k-cores.

Note that the diversified RB-k-core search problem is NP-hard since it can be recognized as the max k-cover problem [15] after obtaining all the RB-k-cores.

Remark. According to the spatial constraint in Definition 3, apparently, if the distance between a vertex v and the query vertex q is larger than $2r$, v cannot be included in any RB-k-cores. We call such vertices faraway vertices, and we can first remove all these vertices from G. We can also safely remove all the vertices that are not in the k-core of G containing q because of the social constraint in Definition 3. We use G_k to denote a connected subgraph of G which is a k-core containing q and for each vertex v in G_k, $d(q, v) \leq 2r$. We use $n = |V(G_k)|$ to represent the number of vertices and $m = |E(G_k)|$ to represent the number of edges of G_k in the following sections. Note that we use Euclidean distance to measure the proximity between two users in this paper, and it can be easily replaced by other measurements (e.g., the geographical distance).

Example 1. Consider a geo-social graph G in Figure 2(a). Suppose Q is the query vertex, given $k = 2$ and $r = 1$, to solve the RB-k-core search problem, we want to find all RB-k-cores from this geo-social graph. We can safely remove vertex A because $d(A, Q) > 2r = 2$ and vertex I because I is not in the 2-core of G. Then we can obtain the candidate geo-social subgraph G_k which is shown in Figure 2(b). As a result, we can find two



(a) Original graph G (b) Candidate graph G_k

Figure 2: An example of the geo-social graph

RB-k-cores $G(S_1)$ and $G(S_2)$, where $S_1 = \{Q, C, J\}$ and $S_2 = \{Q, D, E, F\}$. Furthermore, given $t = 1$, the result of the diversified top-t RB-k-core search problem is S_2.

3 RB-k-CORE SEARCH

In this section, we introduce our algorithms to solve the RB-k-core search problem.

3.1 The Triple-Vertex-Based Algorithm

Firstly we introduce the triple-vertex-based algorithm (TriV) which is designed based on the Exact algorithm in [14]. TriV lies on the following lemma.

Lemma 1. [12] Given a set $S(|S| \geq 2)$ of vertices, the MCC of S can be determined by two or three vertices in S which lie on the boundary of the circle. If two vertices determine it, then the line segment connecting these two vertices must be the circle's diameter. If three vertices determine it, then the triangle consisting of those three vertices is not obtuse.

By Lemma 1, the MCC of a RB-k-core should have two or three vertices lying on its boundary, which are called boundary vertices. Thus, we can enumerate all candidate triple-vertex-combinations and binary-vertex-combinations, then check whether the subgraph enclosed by the circle fixed by the enumerated boundary vertices is a RB-k-core. The details of TriV can be found in [36].

Complexity Analysis. The time complexity of TriV is $O(n^3 \cdot m)$. This is because in TriV, we need to verify $O(n^3)$ candidate triple-vertex-combinations and $O(n^2)$ binary-vertex-combinations. For each combination, we need $O(m)$ time cost to verify the existence of the k-core. Thus, the total time cost of TriV is bounded by $O(n^3 \cdot m)$.

3.2 The Binary-Vertex-Based Algorithm

The major issue of TriV is that we need to verify $O(n^3 + n^2)$ candidate subgraphs based on all triple-vertex- and binary-vertex-combinations. Here we introduce a binary-vertex-based algorithm that only needs to verify $O(n^2)$ candidate subgraphs to solve the RB-k-core search problem.

Based on the definition of RB-k-core, given a query radius r, an obvious observation is that for each RB-k-core in a geo-social graph G, it should be enclosed in at least one circle with radius r. A straightforward approach to finding all RB-k-cores verifies all the circles with radius r in the two-dimensional space. Obviously, there are too many circles with radius r sharing the same RB-k-core in this approach. In other words, for each RB-k-core, we need to ensure that there is at least one circle with radius r enclosing it is checked. This can decrease the number of candidate circles significantly. Firstly, we define the binary-vertex-bounded circle as below.

Definition 5 (Binary-Vertex-Bounded Circle). Given two vertices u and v, we call all circles having u and v lying on the boundary the binary-vertex-bounded circles. A set of binary-vertex-bounded circles with radius r which takes u and v as bounded vertices is denoted as $W_r(u, v)$.

Lemma 2. [17] Given two vertices u and v and a radius r ($r \geq d(u, v)$), we have:

$$|W_r(u,v)| = \begin{cases} 1, \text{iff. } d(u,v) = 2r, \\ 2, \text{iff. } d(u,v) < 2r. \end{cases} \quad (1)$$

Lemma 3. [36] Given a geo-social graph $G(V, E)$, a vertex $q \in V(G)$, a positive integer k, and a query radius r, for each RB-k-core G_k^r, all the vertices in $V(G_k^r)$ should be enclosed in at least one binary-vertex-bounded circle with radius r which takes u and v as the boundary vertices where $u, v \in V(G_k)$.

By Lemma 2, two vertices can bound one/two circles with a given radius r. Based on Lemma 3, we can get all the RB-k-cores in G by verifying all the binary-vertex-bounded circles bounded by vertices in $V(G)$ with radius r. Hence, a more efficient algorithm BinV can be designed by verifying O(n^2) candidate subgraphs constructed from the corresponding binary-vertex-bounded circles, rather than O($n^3 + n^2$) candidate subgraphs as in TriV. The details of BinV can be found in [36].

Complexity Analysis. The time complexity of BinV is $O(n^2 \cdot m)$. This is because, in BinV, we need to verify all the binary-vertex-bounded circles generated from candidate binary-vertex-combinations, which needs $O(n^2 \cdot m)$ time in total.

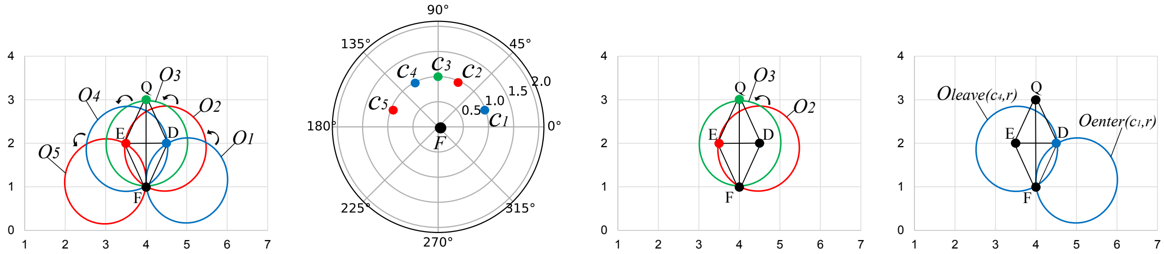
3.3 The Rotating-Circle-Based Algorithms

Although the BinV algorithm improves TriV a lot by reducing the number of candidate subgraphs, it is still not efficient enough. Reviewing the process of BinV, we can observe that the corresponding candidate subgraph is constructed and verified individually for each binary-vertex-combination. There are $O(n^2)$ candidate graphs that need to be constructed, and each of the verification processes takes $O(m)$ time. This motivates us to develop a better algorithm to reduce these candidate subgraphs' construction and verification costs.

In this section, we first present the rotating-circle-based algorithm (RotC), which improves the BinV algorithm by exploring possible cost-sharing in the subgraph construction and verification process. Next, we employ non-trivial pruning techniques to improve the RotC algorithm and propose the optimized rotating-circle-based algorithm (RotC$^+$).

3.3.1 *The Algorithm* RotC

Reviewing Lemma 3, we can find all the RB-k-cores in G by verifying all the candidate subgraphs constructed from corresponding binary-vertex-bounded circles. Considering Example 1, Figure 3(a) is a screenshot of all the binary-vertex-bounded circles which take F as one of the boundary



(a) Circle rotating process (b) Using F as pole (c) Rotating from O_2 to O_3 (d) Entering & leaving circles

Figure 3: An example of Rotating-Circle-Based Algorithms (using vertex F as the pole)

vertices. In the BinV algorithm, we need to verify the candidate graphs enclosed by these binary-vertex-bounded circles one by one. Now we consider putting these binary-vertex-bounded circles into a polar coordinate system using F as the *pole*, and sorting these binary-vertex-bounded circles according to their centers' polar angles. Figure 3(b) shows the centers of binary-vertex-bounded circles in the polar coordinate system, and we can obtain a list of sorted circles $L = \{O_1, O_2, O_3, O_4, O_5\}$. Specifically, in Figure 3(c), O_2 and O_3 are two adjacent binary-vertex-bounded circles. We denote the vertex sets which O_2 and O_3 enclosed as X_2 and X_3, respectively. We can observe that for these two induced subgraphs $G(X_2)$ and $G(X_3)$ where their binary-vertex-bounded circles are adjacent to each other, $V(G(X_2))$ is only one vertex (q) different from $V(G(X_3))$. Based on this observation, we devise a novel algorithm that shares the construction and verification cost for these candidate subgraphs.

In the construction step, we can construct the candidate graphs incrementally after sorting all the binary-vertex-bounded circles. In the verification process, the degree of vertices is easy to maintain dynamically because the difference of enclosed vertices between adjacent binary-vertex-bounded circles is only one vertex. We can divide the binary-vertex-bounded circles into two groups, entering circles and leaving circles. An entering circle denoted as $O_{enter}(c, \gamma)$ is a circle which brings a new vertex in, and a leaving circle $O_{leave}(c, \gamma)$ is a circle which takes an existing vertex out. For example in Figure 3(d), $O_{enter}(c_1, r)$ is an entering circles which brings vertex D in and $O_{leave}(c_4, r)$ is a leaving circle which takes D out of the candidate graph. So for an entering circle, we can avoid recomputing the degree of enclosed vertices when checking the k-core in a binary-vertex-bounded circle. For a leaving circle, we can maintain the degree of vertices and avoid the computation of checking the k-core because there cannot exist a new k-core while a vertex leaves. The detailed rotating-circle-based algorithm (RotC) is shown in Algorithm 1.

In RotC, we first run the core decomposition algorithm and obtain the k-core G_k of G containing q after removing all the faraway vertices in $V(G)$ (line 2). After that, for each vertex v in $V(G_k)$, we set it as the pole in a polar coordinate system P. For each pole v, we generate a candidate vertex set $Y = \{u \in V(G_k) \mid d(u, v) \leq 2r\}$. Then we combine v with other candidate vertices in Y and construct the corresponding binary-vertex-bounded circles based on Lemma 3. We also record whether it is an entering circle or a leaving circle for each binary-vertex-bounded circle. After that, we sort all the binary-vertex-bounded circles in ascending order of their centers' polar angles in P (line 8). Then, for each binary-vertex-bounded circle $O(c, r)$, we compute a set X that contains all the vertices enclosed in O and maintain the degrees of these vertices (lines 9-12). Note that we only need to insert/remove different vertices between O and its precedent binary-vertex-bounded circle, and the degrees of vertices in X can be updated correspondingly. If $O(c, r)$ is an entering circle, we construct a candidate graph $G(X)$, which is a subgraph of G_k induced by X (lines 13-14). After that, we verify whether there exists a k-core containing q in $G(X)$. Because the degrees of vertices in X is already maintained, in the cases such as $deg_{G(X)}(q) < k$, we can skip running a core decomposition to verify the existence of k-core in $G(X)$. Otherwise, if a k-core exists and it satisfies the maximality property, we put the k-core into the result set $\mathcal{R}$ (lines 15-16). Finally, we get all the RB-k-cores in $\mathcal{R}$. As shown in [36], the time complexity of RotC is $O(n^2 \cdot (\log n + m'))$, where $m' \leq m$ and is much smaller than m in practice.

Algorithm 1: RotC

Input: $G(V, E)$: the input graph; q: the query vertex; k, r : constraint parameters
Output: $\mathcal{R}$: a set of RB-k-cores

1 initialize $\mathcal{R} \leftarrow \emptyset$
2 $G_k \leftarrow$ the k-core of G containing q after removing faraway vertices
3 **foreach** *node* $v \in V(G_k)$ **do**
4 $\quad C \leftarrow \emptyset$
5 $\quad$ **foreach** *node* $u \in V(G_k)$ **do**
6 $\quad\quad$ **if** $u \neq v \wedge d(u, v) \leq 2r$ **then**
7 $\quad\quad\quad$ compute $W_r(u, v)$ using $\{u, v\}$ and r
8 $\quad\quad\quad$ put circles in $W_r(u, v)$ into C
9 $\quad$ sort C in ascending order of centers' polar angles
10 $\quad$ **foreach** $O(c, r) \in C$ **do**
11 $\quad\quad$ $X \leftarrow$ a set of vertices enclosed in $O(c, r)$
12 $\quad\quad$ maintain the degree of vertices in X
13 $\quad\quad$ **if** *$O(c, r)$ is an entering circle* **then**
14 $\quad\quad\quad$ construct $G(X)$ from X
15 $\quad\quad\quad$ **if** *exists a G_k^r in $G(X)$* **then**
16 $\quad\quad\quad\quad$ $\mathcal{R}$.update(G_k^r)
17 **return** $\mathcal{R}$

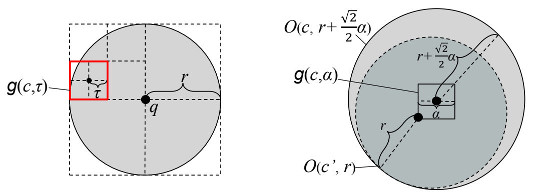
3.3.2 *The Algorithm* RotC$^+$

We continue to introduce the optimized rotating-circle-based algorithm (RotC$^+$), which improves RotC significantly by utilizing novel pruning techniques, including grouping-based pre-process and the in-process pruning rules.

The Pre-Process Pruning. Firstly, we introduce the grouping-based pre-process pruning technique to partition the vertices into groups and filter out unpromising candidate vertices. The pruning technique is based on the following lemma.

Lemma 4. Given a geo-social graph $G(V,E)$, a vertex $q \in V(G)$, a positive integer k and a query radius r, for each RB-k-core G_k^r in G, the center point c of the MCC $O(c,\gamma)$ of $V(G_k^r)$ should satisfy $d(c,q) \leq r$.

Proof. By Definition 3, for a RB-k-core G_k^r, the MCC $O(c,\gamma)$ of G_k^r should enclose q and satisfy $\gamma \leq r$. Hence we have $d(c,q) \leq r$ and complete the proof. ■



(a) Grouping $O(q,r)$, $\tau = 2$ (b) Verifying a group $g(c,\alpha)$

Figure 4: An example of grouping-based pre-process

Lemma 4 illustrates that all the centers of MCCs of RB-k-cores are in the circle $O(q,r)$. Apparently, the circle $O(q,r)$ can be partitioned into four groups which are squares with size $r \times r$. Similarly, the group with size $r \times r$ can also be partitioned into 4 smaller groups with size $\frac{r}{2} \times \frac{r}{2}$. Hence, given a grouping parameter τ, as shown in Figure 4(a), we can partition the circle from 4 groups with size $r \times r$ to $4\lceil \frac{r}{\tau} \rceil^2$ groups with size $\tau \times \tau$ iteratively. In each iteration, we halve the group size and prune the groups which do not need further verification. For example, in Figure 4(a), if we set $\tau = \frac{r}{2}$, the pre-process will run 2 iterations in total and in each iteration, we need to verify at most 4 and 16 groups with size = r and $\frac{r}{2}$, respectively.

We proceed to present the verification process of a given group of vertices denoted as $g(c,\alpha)$, where c is the center point and α is the side length. As shown in Figure 4(b), because the longest distance between c and the other points in $g(c,\alpha)$ is $\frac{\sqrt{2}}{2}\alpha$, we can use the circle $O(c, r+\frac{\sqrt{2}}{2}\alpha)$ to enclose all the circles with radius r and centered at a point in $g(c,\alpha)$. In other words, for each circle $O(c',r)$ which centers at $g(c,\alpha)$ as shown in Figure 4(b), $O(c, r+\frac{\sqrt{2}}{2}\alpha)$ can enclose it. Then, we can construct an induced subgraph $G(X)$ of G using X which contains all the vertices enclosed in the circle $O(c, r+\frac{\sqrt{2}}{2}\alpha)$.If there exists no k-core containing q in $G(X)$, we can prune the whole group $g(c,\alpha)$. Otherwise, if the MCC $O(c',\alpha')$ of the k-core $G(X)_k$ containing q has the radius $\alpha' \leq r$, we can mark $G(X)_k$ as a candidate result and prune the whole group $g(c,\alpha)$, because $G(X)_k$ is the only result that can be found using the vertices in $g(c,\alpha)$. Otherwise, if the MCC $O(c',\alpha')$ of $G(X)_k$ has the radius $\alpha' > r$, the RB-k-cores obtained from $g(c,\alpha)$ are subsets of $G(X)_k$. Thus, we add the vertices in $G(X)_k$ into a candidate vertex set and do further check for the group $g(c,\alpha)$. The details of the pre-processing is shown in Algorithm 2.

Algorithm 2: Grouping-Based Pre-Process

Input: $G(V,E)$: the input graph; q: the query vertex; k, r: constraint parameters; τ: grouping parameter; $\mathcal{R}$: candidate result set
Output: G_k: a graph

```
α ← r; Y ← g(q, 2r)
G_k ← the k-core of G containing q after removing
  faraway vertices
while α ≥ τ do
  foreach group g(c, 2α) ∈ Y do
    partition g(c, 2α) into four groups with size
      α × α and put then into Y_c
  Y ← ∅; S ← ∅
  foreach group g(c, α) ∈ Y_c do
    construct graph G(X) using X which contains
      vertices enclosed in O(c, r + (√2/2)α)
    if exists a G_k^r in G(X) then
      R.update(G_k^r)
    else if exists a k-core G(X)_k in G(X) then
      Y.insert(g)
      put vertices in V(G(X)_k) into S
  foreach node v ∈ V(G_k) do
    if v ∉ S then
      remove vertex v from V(G_k)
  α = α/2
return G_k
```

In-Process Pruning Techniques. During finding RB-k-cores from G_k, we introduce two in-process pruning rules to push the efficiency boundary further.

Pruning Rule 1: Overall Checking. Reviewing the process of the RotC algorithm, we choose a vertex v from $V(G_k)$ as the pole and generate a candidate vertex set $S = \{u \in V(G_k) \mid d(u,v) \leq 2r\}$. Then we construct an induced subgraph $G(S)$ using vertices in S and compute the k-core $G(S)_k$ of $G(S)$ containing q. If $G(S)_k$ does not exist or the vertices in $V(G(S)_k)$ are all enclosed in the MCC of a candidate RB-k-core in $\mathcal{R}$, we can prune the pole v.

Pruning Rule 2: Circle Filtering. In the RotC algorithm, after choosing the pole v and corresponding candidate vertices, we combine v with all candidate vertices and generate the binary-vertex-bounded circles.

Firstly, we can prune all the circles which exclude the query vertex q. After that, because there is only one vertex difference between two adjacent circles, we can compute the vertex difference between a circle and its precedent. We divide the circles into two groups, the entering circles and leaving circles, respectively. For each entering circle, we record the vertex it brings in, and for each leaving circle, we record the vertex it moves out. For the group of entering circles, we sort them in ascending order of their centers' polar angles and put them into a list L_{enter}. Then for each entering circle O_{enter} in L_{enter}, we compute a vertex set $\mathcal{V}(O_{enter})$ which contains all the vertices bringing from the entering circles that appear before O_{enter} in L_{enter}. This can be done by incrementally adding the vertices bringing from the first entering circle to the last entering circle, and the time complexity is $O(L_{enter})$. It is obvious that the number of vertices in $\mathcal{V}(O_{enter})$ monotonously increases with the index of O_{enter} in L_{enter}. Thus, we can use binary search to find the first entering circle O'_{enter} in L_{enter} such that we can construct a k-core from $\mathcal{V}(O'_{enter})$ containing q.

Algorithm 3: DivBS

Input: $G(V, E)$: the input graph; q: the query vertex; k, t, r: constraint parameters; τ: grouping parameter
Output: $\mathcal{D}$: a set of diversified RB-k-cores

```
1  R ← RotC+(G, q, k, r, τ)
2  initialize D ← ∅
3  initialize V* ← ∅
4  foreach B ∈ R do
5  |   V* ← V* ∪ V(B)
6  for i = 1 ... t do
7  |   cov_max ← 0
8  |   B* ← ∅
9  |   foreach B ∈ R do
10 |   |   if |B ∩ V*| > cov_max then
11 |   |   |   B* ← B
12 |   |   |   cov_max ← |B ∩ V*|
13 |   V* ← V* \ V(B*)
14 |   D ← D ∪ B*
15 return D
```

The circles appearing before O'_{enter} can be safely discarded because they cannot contain a RB-k-core. Similarly, we sort in descending order of their centers' polar angles for all the leaving circles and put them into L_{leave}. In the same way, we can find the first leaving circle O'_{leave} such that we can construct a k-core from $\mathcal{V}(O'_{leave})$ containing q and discard all the circles before O'_{leave} in L_{leave}. In this way, we can reduce the number of binary-vertex-bounded circles, which need to be verified in the next stage.

The details of RotC$^+$ with the above pruning techniques can be found in [36]. Note that the time complexity of RotC$^+$ is $O(\lceil\frac{r}{\tau}\rceil^2 \cdot m \cdot (\log(\lceil\frac{r}{\tau}\rceil)+1) + |F| \cdot m + |F_1| \cdot \log|F_1| \cdot (|F_1| + m) + |F_1| \cdot |F_2| \cdot m')$, where F denotes the candidate vertex set after pre-process pruning ($|F| \leq n$), F_1 is the vertex set obtained from F after the overall checking, F_2 is the set of circles that need to be verified ($|F_2| \leq n$), and m' is the average time cost of verifying the existence of a k-core ($m' \leq m$) [36].

4 Diversified RB-k-core Search

In the above section, we study how to find all the RB-k-cores efficiently. However, in many real-world applications, we only need to retrieve representative RB-k-cores for analysis. Motivated by this, we study how to solve the diversified (top-t) RB-k-core search problem in this section.

4.1 A Baseline Algorithm

As presented in Section 3.3, The algorithms RotC, and RotC$^+$ can find all the RB-k-cores efficiently. Based on these algorithms, an algorithm DivBS to solve the diversified RB-k-core search problem can be naturally devised. Firstly, DivBS obtains all the RB-k-cores using the RotC$^+$ algorithm, and then the diversified RB-k-core search problem becomes the max k-cover problem, which can be solved using a greedy approach with a guaranteed approximation ratio [15].

The details of the DivBS algorithm are shown in Algorithm 3. Firstly, the DivBS algorithm runs the algorithm RotC$^+$ to get the result set $\mathcal{R}$, which contains all the RB-k-cores. After that, it processes each RB-k-core in $\mathcal{R}$ and puts the vertices in RB-k-core into V^*. Note that V^* contains all the different vertices covered by the RB-k-cores in $\mathcal{R}$ (lines 4-5). Then, DivBS runs t iterations, and for each iteration, we get the RB-k-core B^* which covers the most number of uncovered vertices (lines 6-14). After running lines 6 - 14, $\mathcal{D}$ is an approximate result of the set of diversified top-t RB-k-cores. Note that, the DivBS algorithm achieves the same approximation ratio (i.e., $(1-1/e) \approx 0.632$) as the best possible polynomial-time approximation algorithm for the max k-cover problem [15].

Time Complexity of DivBS**.** Firstly, DivBS needs $O(T_{\mathsf{RotC^+}})$ time to get all the RB-k-cores where $T_{\mathsf{RotC^+}}$ is the time complexity of RotC$^+$. After that, DivBS needs $O(t \cdot \sum_{B\in\mathcal{R}} |V(B)| \cdot cov(\mathcal{R}))$ time to get the diversified top-t RB-k-cores. This is because DivBS need to run t iterations and in each iteration, we need to compute cov_{max} using $O(\sum_{B\in\mathcal{R}} |V(B)| \cdot cov(\mathcal{R}))$ time. Totally, the time complexity of DivBS is $O(T_{\mathsf{RotC^+}} + t \cdot \sum_{B\in\mathcal{R}} |V(B)| \cdot cov(\mathcal{R}))$.

4.2 Maintenance-Based Solutions

Motivation. Although DivBS achieves a good approximation ratio, it isolates the RB-k-core finding process from the diversified top-t RB-k-core search process. Thus, DivBS cannot have much pruning ability when finding the RB-k-core, and it keeps all the RB-k-cores in memory. Motivated by the above observations, we devise new algorithms that can maintain the diversified top-t candidates with the following advantages. Firstly, the new algorithms only need to maintain t candidate RB-k-cores in $\mathcal{D}$ rather than keeping all the RB-k-cores in the memory and checking for the diversification. Secondly, they have a higher pruning ability than DivBS. As the diversification checking process is integrated into finding RB-k-cores, we can develop pruning techniques to early terminate unpromising processing and reduce the searching space. Thirdly, the new algorithms can achieve a bounded approximation ratio of 0.25, as shown in the algorithm analysis.

4.2.1 *The Algorithm* DivRotC$^+$

Before introducing the new algorithm, we first define the private-vertex-coverage of a RB-k-core.

Definition 6 (Private-Vertex-Coverage). Given a set of RB-k-cores $\mathcal{D} = \{B_1, B_2, ...\}$ in G, the private-vertex-coverage of each $B \in \mathcal{D}$ denoted by $pvcov(B)$ is the set of vertices in B that are not covered by the other RB-k-cores in $\mathcal{D}$, i.e., $pvcov(B, \mathcal{D}) = V(B) \backslash cov(\mathcal{D} \backslash B)$.

Based on the definition of private-vertex-coverage, we can have the following definition of Min-cover RB-k-core.

Definition 7 (Min-Cover RB-k-core). Given a set of RB-k-cores $\mathcal{D} = \{B_1, B_2, ...\}$ in G, the min-cover RB-k-core of $\mathcal{D}$, denoted by $B_{min}(\mathcal{D})$, is the RB-k-core $B \in \mathcal{D}$ with the smallest $pvcov(B, \mathcal{D})$, i.e., $B_{min}(\mathcal{D}) = \arg\min_{B\in\mathcal{D}} |pvcov(B, \mathcal{D})|$.

With the definition of Min-cover RB-k-core, we propose the DivRotC$^+$ algorithm. DivRotC$^+$ is based on RotC$^+$, and a new result updating function is used on RotC$^+$ to ensure only t RB-k-cores are maintained in $\mathcal{D}$. We show the details of DivRotC$^+$ in Algorithm 4. We first initialize the result set $\mathcal{D}$ and run the grouping-based pre-process (lines 1-2). In the grouping-based pre-processing, the updating of $\mathcal{R}$ is replaced with the updating of $\mathcal{D}$. In addition, since candidate RB-k-cores can be retrieved in this process, we replace the update process in line 10 with a new updating function **DivUpdate**. After that, for each vertex v in $V(G_k)$, we set it as the pole in a polar coordinate system P (line 3). For each pole v, we generate candidate binary-vertex-bounded circles and corresponding candidate graphs following the same framework as RotC$^+$ (lines 4-21). For each candidate subgraph, if there exists a RB-k-core G_k^r in it, we run the function **DivUpdate**$(G_k^r, \mathcal{D})$ to check whether $\mathcal{D}$ can be updated (lines 22-24). In the function **DivUpdate**, we directly insert G_k^r into $\mathcal{D}$ if $|\mathcal{D}| < t$. Otherwise, we generate $\mathcal{D}'$ by replacing $B_{min}(\mathcal{D})$ with G_k^r. If G_k^r meets the condition that $|pvcov(G_k^r, \mathcal{D}')| > \delta$, we replace $\mathcal{D}$ with $\mathcal{D}'$. Here δ is the updating threshold which is equal to $|pvcov(B_{min}(\mathcal{D}), \mathcal{D})| + \frac{cov(\mathcal{D})}{t}$. Finally, we get a set of RB-k-cores in $\mathcal{D}$.

Threshold maintenance for early termination. Deriving from the candidate updating condition, we can have the following lemma.

Lemma 5. Given a geo-social graph $G(V, E)$, a subgraph G' of G, and a set of t candidate RB-k-cores in $\mathcal{D}$, there is no RB-k-core in G' can be included in $\mathcal{D}$ by Algorithm DivRotC$^+$ if $|V(G')| \leq \delta$.

Proof. According to the candidate update process in Algorithm 4 lines 27-33, a RB-k-core G_k^r can be included in $\mathcal{D}$ if the private-vertex-coverage after replacing $B_{min}(\mathcal{D})$ with G_k^r is increased by more than $\frac{cov(\mathcal{D})}{t}$. Since adding G_k^r increases the private-vertex-coverage by at most $|G_k^r|$, we can get that if $G_k^r \in V(G')$, the private-vertex-coverage cannot be increased by more than $|V(G')|$. Thus, this lemma holds. ■

According to lemma 5, we can maintain the threshold δ each time we update the candidate RB-k-cores in $\mathcal{D}$. After that, a candidate subgraph in the finding process can be pruned directly if the size of it is less than δ. In Algorithm 4, the early termination checking according to the threshold can be applied in the following stages: 1) checking the size of $G(X)$ in line 8 in the grouping-based pre-process (i.e., Algorithm 2); 2) checking the size of $G(X)$ in line 12; 3) checking the number of vertices in X in line 19. Note that the number of vertices in X can be dynamically maintained since there is at most one vertex is included/excluded between adjacent circles.

Analysis of DivRotC$^+$**.** We first show the theoretical guarantee of the result of Algorithm 4, and then the time complexity of Algorithm 4.

Theoretical guarantee of the result. Suppose $\mathcal{D}$ is the result provided by Algorithm 4 and $\mathcal{D}^*$ is the set of optimal diversified top-t RB-k-cores. We have $|cov(\mathcal{D})| \geq 0.25 \times |cov(\mathcal{D}^*)|$. This can be easily extended by the theoretical result shown in [4] which analyses an online approximate algorithm for maximum k-coverage problem.

Time complexity. The time complexity of DivRotC$^+$ is $O(T_{\text{RotC}^+} + \sum_{B \in \mathcal{R}} (|V(B)| + |V(B_{max})|))$. Here, T_{RotC^+} is the time complexity of RotC$^+$, $\mathcal{R}$ is the set of all the RB-k-cores, and B_{max} denotes the RB-k-core with the largest number of vertices in $\mathcal{R}$. Note that, the second term is the time complexity of candidate updating. For each $B \in \mathcal{R}$, we need to compute the private-vertex-coverage after replacing $B_{min}(\mathcal{D})$ with B which needs $O(|V(B)| + |V(B_{max})|)$ time.

Algorithm 4: DivRotC$^+$

Input: $G(V, E)$: the input graph; q: the query vertex; k, r: constraint parameters; τ: grouping parameter
Output: $\mathcal{D}$: a set of RB-k-cores

1 initialize $\mathcal{D} \leftarrow \emptyset$
2 $G_k \leftarrow PreProcess(G, q, k, r, \tau, \mathcal{D})$, replace $\mathcal{R}$ with $\mathcal{D}$, update $\mathcal{D}$ using function **DivUpdate** in line 10, early terminate the processing of X if $|X| \leq \delta$
3 **foreach** *node* $v \in V(G_k)$ **do**
4   $C \leftarrow \emptyset$; $X \leftarrow \emptyset$
5   **foreach** *node* $u \in V(G_k)$ **do**
6   **if** $u \neq v \wedge d(u, v) \leq 2r$ **then**
7   put u into X
8   compute $W_r(u, v)$ using $\{u, v\}$ and r
9   put circles in $W_r(u, v)$ into C
10   **if OverallChecking(**X**) = false then**
11   continue
12   **if** $|\mathcal{D}| > t \wedge |X| \leq \delta$ **then**
13   continue
14   sort C in ascending order of centers' polar angles
15   employ circle filtering to C
16   **foreach** $O(c, r) \in C$ **do**
17     $X \leftarrow$ a set of vertices enclosed in $O(c, r)$
18   maintain the degree of vertices in X
19   **if** $|\mathcal{D}| > t \wedge |X| \leq \delta$ **then**
20   continue
21   **if** $O(c, r)$ *is an entering circle* **then**
22   construct $G(X)$ from X
23   **if** *exists a* G_k^r *in* $G(X)$ **then**
24   **DivUpdate(**G_k^r, $\mathcal{D}$**)**
25 **return** $\mathcal{D}$
26
27 **DivUpdate(**G_k^r, $\mathcal{D}$**)**
28 **if** $|\mathcal{D}| < t$ **then**
29   $\mathcal{D} \leftarrow \mathcal{D} \cup G_k^r$
30 **else**
31   $\mathcal{D}' \leftarrow (\mathcal{D} \backslash B_{min}(\mathcal{D})) \cup G_k^r$
32   **if** $|pvcov(G_k^r, \mathcal{D}')| > |pvcov(B_{min}(\mathcal{D}), \mathcal{D})| + \frac{cov(\mathcal{D})}{t}$ **then**
33     $\mathcal{D} \leftarrow \mathcal{D}'$

4.2.2 *The Algorithm* AdvDivRotC$^+$

Following the framework of DivRotC$^+$, here we propose optimization techniques to improve the efficiency of DivRotC$^+$. Specifically, we first propose a tighter upper bound for the private-vertex-coverage of candidate subgraphs. Secondly, we explore the effect of the processing order of poles.

A tighter upper bound. In Lemma 5, we use $|V(G')|$ as the upper bound of the private-vertex-coverage increasing. Although $|V(G')|$ is a correct upper bound, it cannot reflect the number of private vertices in the possible RB-k-cores in the candidate subgraph G' and may not be very tight. In order to obtain a tighter upper bound, we maintain a $O(n)$ hash table J, which contains all the vertices in $\mathcal{D} \backslash B_{min}(\mathcal{D})$. When we need to check a candidate subgraph G', we obtain the upper bound of the number of uncovered vertices provided by G' by computing $|V(G') \backslash V(J)|$. Then, we can early terminate the checking if $|V(G')\backslash V(J)| \leq \delta$. Note that the value $|V(G')\backslash V(J)|$ can be easily computed in $O(V(G'))$ time, and it can also be dynamically maintained when processing the circles in line 15 in Algorithm 4.

The processing order of poles. Apart from the above bound, we can also consider the processing order of poles. This is because a result set with large private-vertex-coverage can increase the chance to prune the unpromising candidate subgraphs. We rank the candidate poles according to the following equation and choose the pole with the highest rank in each iteration.

$$rank(v) = \alpha(v) + \frac{|V(G^*)\backslash V(J)|}{|V(G_k)|} \quad (2)$$

Here, we use $\alpha(v)$ to ensure that we first process the pole which is not in the candidate result set. We set $\alpha(v) = 0$ if the vertex v is contained in a RB-k-core in the current result set $\mathcal{D}$. Otherwise, we set $\alpha(v) = 1$. Secondly, we prefer the pole which may expand more vertices which are not covered by the RB-k-cores in the current result set. We use G^* to estimate this term. For each $u \in V(G^*)$, it should satisfy the following constraints: 1) u is a neighbor of v; 2) $d(u,v) \leq r$; Thus, we compute the second term of $rank(v)$ by $\frac{|V(G^*)\backslash V(J)|}{|V(G_k)|}$. Here, G_k is the candidate subgraph after running the pre-processing and J is the set of vertices in $\mathcal{D}\backslash B_{min}(\mathcal{D})$. Note that, for each vertex v, $\alpha(v)$ can be obtained in $O(cov(\mathcal{D}))$ time. G^* can be obtained in $O(deg_G(v))$ time and we can compute $\frac{|V(G^*)\backslash V(J)|}{|V(G_k)|}$ in $O(G^*)$ time.

The AdvDivRotC$^+$ algorithm. Utilizing the above strategies, we propose the AdvDivRotC$^+$ algorithm as shown in Algorithm 5. We first initialize the result set $\mathcal{D}$ and run the grouping-based pre-processing (lines 1-2). In the grouping-based pre-processing, comparing with DivRotC$^+$, we use advanced early termination condition after line 8 in Algorithm 2. After that, for each vertex v in $V(G_k)$ with the highest $rank(v)$, we set it as the pole in a polar coordinate system P (line 3). The rank value is computed according to Equation 2. For each pole v, we first get all the candidate vertices and check whether we can skip the process of v. Then, we generate candidate binary-vertex-bounded circles and corresponding candidate subgraphs following the same framework as DivRotC$^+$. For each candidate subgraph, if it satisfies the upper bound checking and there exists a RB-k-core G_k^r in it, we run the function **AdvDivUpdate**$(G_k^r, \mathcal{D})$ to check whether $\mathcal{D}$ can be updated. In the function **AdvDivUpdate**, we directly insert G_k^r into $\mathcal{D}$ if $|\mathcal{D}| < t$. Otherwise, we generate $\mathcal{D}'$ by replacing $B_{min}(\mathcal{D})$ with G_k^r. If G_k^r meets the condition that $|pvcov(G_k^r, \mathcal{D}')| > |pvcov(B_{min}(\mathcal{D}), \mathcal{D})| + \frac{cov(\mathcal{D})}{t}$, we replace $\mathcal{D}$ with $\mathcal{D}'$. Finally, we get a set of RB-k-cores in $\mathcal{D}$.

Algorithm 5: AdvDivRotC$^+$

Input: $G(V,E)$: the input graph; q: the query vertex; k, r: constraint parameters; τ: grouping parameter
Output: $\mathcal{D}$: a set of RB-k-cores

```
1  initialize D ← ∅
2  G_k ← PreProcess(G, q, k, r, τ, D), replace R with D,
     early terminate the processing of X if
     |D| > t ∧ X\V(J) ≤ δ, update D using function
     AdvDivUpdate in line 10
3  foreach node v ∈ V(G_k) with the highest rank do
4      C ← ∅; S ← ∅; J ← ∅; δ = 0
5      run Algorithm 5 lines 5-24, replace the condition
         (|D| > t ∧ X ≤ δ) with (|D| > t ∧ X\V(J) ≤ δ) in
         lines 12 and 19;
6  return D
7
8  AdvDivUpdate(G_k^r, D)
9  if |D| < t then
10     D ← D ∪ G_k^r
11 else
12     D' ← (D\B_min(D)) ∪ G_k^r
13     if |pvcov(G_k^r, D')| > |pvcov(B_min(D), D)| + cov(D)/t
         then
14         D ← D'
15         δ ← |pvcov(B_min(D), D)| + cov(D)/t
16         J ← V(D\B_min(D))
```

5 EXPERIMENTS

In this section, we report the evaluation of the effectiveness of our model and the efficiency of our algorithms.

5.1 Experimental Settings

Algorithms. In the experimental study, we implement and evaluate four algorithms to solve the RB-k-core search problem: the triple-vertex-based algorithm TriV in Section 3.1, the binary-vertex-based algorithm BinV in Section 3.2, the rotating-circle-based algorithm RotC in Section 3.3, and the optimized rotating-circle-based algorithm RotC$^+$ in Section 3.3. We also extend our RotC$^+$ algorithm to solve the SAC (spatial-aware community) search problem proposed in [14]. We also evaluate the algorithms DivBS, DivRotC$^+$ and AdvDivRotC$^+$ to solve the diversified top-t RB-k-core search problem in Section 4.

The algorithms are implemented in C++, and the experiments are run on a Linux server with Intel Xeon E5-2687W (3.4GHz, 8 Cores) processor and 64GB main memory. We randomly select 200 query vertices and report the average result for these queries. We terminate an algorithm if the running time is more than three hours.

Table 2: Summary of Datasets

Dataset	$\|V\|$	$\|E\|$	d_{avg}
Brightkite	51,406	197,167	7.67
Gowalla	107,092	456,830	8.53
Flickr	214,698	2,096,306	19.5
Foursquare	2,127,093	8,640,352	8.12
Synthetic	4,000,000	40,000,000	20

Datasets. We use four real datasets in our experiments including `Brightkite`, `Gowalla`, `Flickr`, and `Foursquare`. In the four datasets, we consider each user associated with a geo-location coordinate (latitude and longitude) as a vertex and the friendship between two users as an edge. Helmert transformation [37] is adopted to transform geo-location coordinates of vertices to Cartesian coordinates.

We also conduct experiments on a synthetic dataset `Synthetic`. We first generate a non-spatial graph using a well-known graph generator GTGraph [1]. The degrees of the vertices in the graph follow a power-law distribution, as often used in the study of social networks. After generating the graph, we randomly generate the locations of the vertices in a square of side 300km.

Parameters. The experiments are conducted using different settings on 4 parameters: k (the minimum degree), r (the maximal radius), τ (the parameter used in the pre-process of RotC$^+$), and n (the percentage of vertices). We vary k from 4 to 16 and set 4 as the default value. We vary r from 1km to 40km and set r to 5km by default. When varying the graph size, we randomly sample 20% to 100% vertices of the original graphs and construct the induced subgraphs using these vertices. The parameter n is varied from 20% to 100%, representing the percentage of the vertices we use in each dataset. The parameter τ is varied from r to $\frac{r}{16}$ which controls the number of iterations of the pre-processing in RotC$^+$. We vary t, which is a parameter used to control the number of diversified top-t RB-k-cores from 1 to 9 and set 5 as the default value.

5.2 Effectiveness Evaluation

In this section, we show the effectiveness of our RB-k-core model and the diversified RB-k-core model.

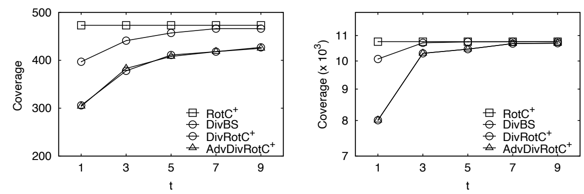
(a) Case study on Gowalla (q=1396, k=3, r=0.76km) (b) Case study on Flickr (q=111419, k=3, r=1.67km)

Figure 5: Case studies

Case study. We present two case studies to show the result of RB-k-core search on `Gowalla` and `Flickr` in Figure 5(a) and Figure 5(b), respectively. The query vertices are marked by question mark symbols. Under setting q=1396, k = 4 and r = 0.76km on `Gowalla`, we can get two RB-k-cores containing q as shown in Figure 5(a). We mark the vertices and the MCC of these two RB-k-cores in black color and grey color, respectively. We can see that, the social constraint and the spatial constraint both contribute to the construction of these two RB-k-cores. For example, if the social constraint is ignored, the black vertices enclosed by the grey circle will be included in the grey RB-k-core. On the other hand, all the vertices in Figure 5(a) will be united into one community if the radius constraint is not being considered. Figure 5(b) shows the result of RB-k-core search on `Flickr` using q = 111419, k = 3 and r = 1.67km contains two retrieved communities. Using the same q and k, the SAC search (i.e., a similar model in [14]) will provide the communities with black color as shown in Figure 5(a) and Figure 5(b). The radiuses of the black circles are 0.74km and 1.67km in Figure 5(a) and Figure 5(b), respectively. Compared to the SAC search, in Figure 5(a), our RB-k-core search can give users more options by slightly increasing the minimum radius (i.e., from 0.74km to 0.76km) for the same q and k. In Figure 5(b), the RB-k-core search is able to provide users more than one selection for the same q, k, and minimum r.

1. **http://www.cse.psu.edu/~kxm85/software/GTgraph/**

Effectiveness of the diversified RB-k-core model. Here, we show the effectiveness of the diversified RB-k-core search by comparing the total coverage of the result sets provided by RotC$^+$, DivBS, DivRotC$^+$, and AdvDivRotC$^+$. We vary the parameter t, which controls the number of different RB-k-cores returned by the algorithms.



(a) Gowalla, varying t (b) Foursquare, varying t

Figure 6: Evaluate the total coverage

We can see that in Figure 6, the total coverage increases when t increases for all the algorithms DivBS, DivRotC$^+$, and AdvDivRotC$^+$. This is obvious since the number of RB-k-cores in the result sets increases when t increases. We can also observe that the coverages of the algorithms DivBS, DivRotC$^+$, and AdvDivRotC$^+$ are only slightly smaller than the RotC$^+$ algorithm, especially when $t > 1$. This validates the effectiveness of our diversified RB-k-core model. In addition, the total coverage of AdvDivRotC$^+$ is smaller than DivBS. This is because AdvDivRotC$^+$ maintains only top-t RB-k-cores in the computation and is more efficient.

5.3 Efficiency Evaluation of the algorithms to solve the RB-k-core search problem

In this section, we first evaluate the efficiency of the proposed four algorithms to solve the RB-k-core search problem on all the datasets. Then we evaluate the effect of k and the scalability of the proposed algorithm. After that, the parameter τ used in the RotC$^+$ algorithm is evaluated. Finally, we extend our RotC$^+$ algorithm to solve the SAC search problem [14] and compare the performance.

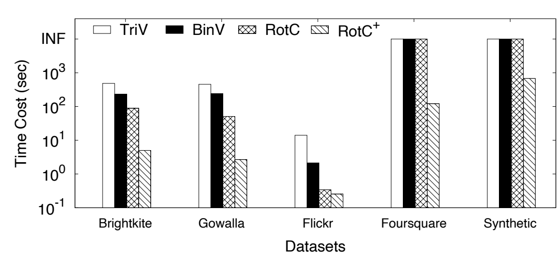
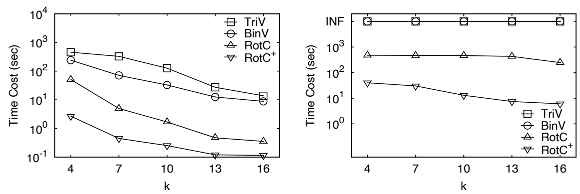


Figure 7: Performance on different datasets

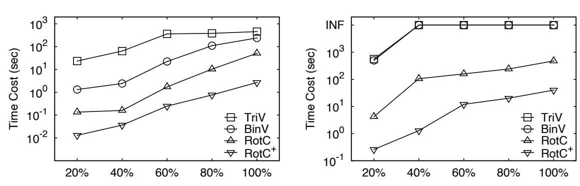
Evaluating the performance of all algorithms on different datasets. In Figure 7, we show the performance of our RB-k-core search algorithms on five datasets. We set k as default and r to 1km, 5km, 10km, 20km, 40km on `Brightkite`, `Gowalla`, `Flickr`, `Foursquare` and `Synthetic`, respectively. In the meantime, we keep the other parameters fixed to default values. We can observe that BinV is more efficient than TriV on `Brightkite`, `Gowalla`, and `Flickr`. The algorithms RotC and RotC$^+$ using the rotating circle strategy are more efficient than TriV and BinV on the three datasets because RotC and RotC$^+$ can compute the RB-k-cores in an incremental manner, which significantly reduces the computation cost. On `Foursquare` and `Synthetic`, we can see that only RotC$^+$ can return the results within the time limit. `Foursquare` is much larger than the first three datasets, which means many more candidate vertices to be processed. On `Synthetic`, the vertices are more densely distributed over the space than the other datasets, and thus the candidate circles contain more vertices. In summary, as shown in Figure 7, our RotC$^+$ algorithm significantly outperforms the other three algorithms on all datasets.



(a) Gowalla, varying k (b) Foursquare, varying k

Figure 8: Effect of k

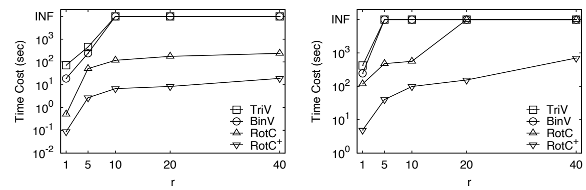
Evaluating the effect of k. Figure 8 evaluates the effect of k for four algorithms on `Gowalla` and `Foursquare`. We vary k from 4 to 16 and fix the other parameters as default values. In Figure 8(a), we can observe that the time cost of all four algorithms drops when k increases because of the number of vertices in the k-core of the original graph (selected as candidate vertices) decreases. Similar trends can be observed in Figure 8(b). As expected, RotC and RotC$^+$ significantly outperform TriV and BinV on both datasets due to the usage of the rotating circle technique. For example, on both datasets, RotC is about one order of magnitude faster than TriV and BinV, and RotC$^+$ is at least two orders of magnitude faster than TriV and BinV.



(a) Gowalla, varying n (b) Foursquare, varying n

Figure 9: Effect of graph size

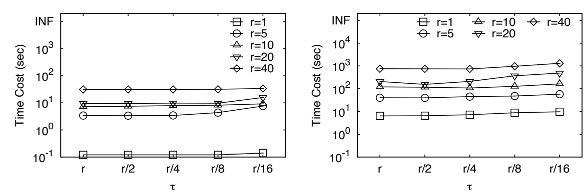
Scalability. (1) *Evaluating the effect of graph size.* Figure 9 shows the scalability of four algorithms by varying the graph size from 20% to 100% in all datasets. We can observe that, on `Gowalla`, all these four algorithms are scalable and their running time increases as the percentage of vertices increases. As shown in Figure 9(b), on `Foursquare`, TriV and BinV can only finish within the time limit when n=20%, while RotC and RotC$^+$ have similar trends as in Figure 9(a). As discussed before, RotC$^+$ is more efficient than the other three algorithms.



(a) Gowalla, varying r (b) Foursquare, varying r

Figure 10: Effect of r

(2) *Evaluating the effect of r.* Figure 10 illustrates the effect of r on `Gowalla` and `Foursquare`. We vary r from 1km to 40km and fix the other parameters as default values. In Figures 10(a) and 10(b), the time cost increases as r becomes larger because the number of vertices in circle $O(q, 2r)$ grows when r increases. We can also see that, on `Gowalla`, both RotC and RotC$^+$ are several orders of magnitude faster than TriV and BinV. On `Foursquare`, TriV and BinV can only compute the result when r = 1km and RotC can get the results when r is no more than 10km within reasonable time. As expected, the RotC$^+$ algorithm significantly outperforms the other three algorithms on `Foursquare`, and the time cost is stable when r is large on both datasets.



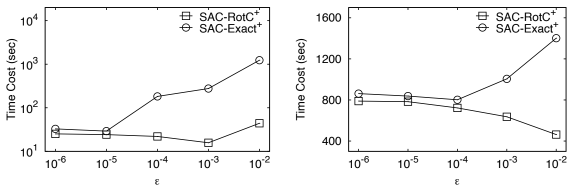
(a) Gowalla, varying τ (b) Foursquare, varying τ

Figure 11: Effect of τ

s

Evaluating the effect of τ. Figure 11 illustrates the effect of τ, which is a parameter used in the grouping-based pre-processing in RotC$^+$. Because the value of τ is related to r, we set r to 1km, 5km, 10km, 20km, and 40km on both `Gowalla` and `Foursquare`. As discussed before, as τ increases, the time cost of pre-processing increases, and the number of candidate vertices decreases. We can observe that the running time is not very sensitive to τ when τ is relatively large on the two datasets. The time cost starts to increase from $\tau = \frac{r}{4}$ in most cases because the number of vertices that can be pruned increases slowly, and the time cost of pre-processing begins to dominate the cost of RotC$^+$. Hence we set $\tau = \frac{r}{4}$ in our experiments on all datasets.

Extend to solve the SAC search problem [14]. As discussed before, the SAC search problem can be solved by slightly modifying our RotC$^+$ algorithm using the binary search. In Figure 12, we study the performance of the SAC-RotC$^+$ algorithm, which is extended from RotC$^+$ to solve the SAC search problem, and we compare its performance with the state-of-the-art exact algorithm SAC-Exact$^+$ proposed in



(a) Gowalla, varying ϵ (b) Foursquare, varying ϵ

Figure 12: Extend to solve SAC search problem

[14]. Fang et al. [14] implemented the SAC-Exact$^+$ algorithm in JAVA, while we implement the SAC-Exact$^+$ algorithm in C++ for the fairness of comparison.

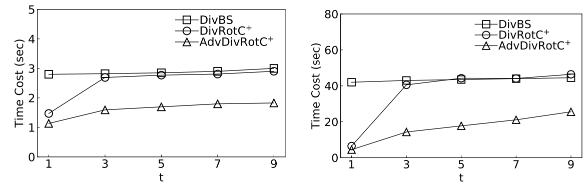
The SAC-Exact$^+$ algorithm includes two phases. Firstly, it conducts the quad-tree-based vertex pruning phase, which can reduce the number of potential vertices. Next, it conducts a triple-vertex-based algorithm, which is similar to the TriV algorithm in this paper. In the RB-k-core search problem, we have analyzed that the triple-vertex-based algorithm is time-consuming, and it can be improved by the rotating circle strategy to compute the result incrementally. We can do the same thing in the SAC search problem. In our SAC-RotC$^+$ algorithm, we also conduct the vertex pruning phase, but we adopt the rotating-circle-based algorithm in the second phase. Note that the in-process pruning technique in RotC$^+$ can also be applied in SAC-RotC$^+$, but the pre-process pruning technique cannot be used because of the model difference.

We vary the parameter ϵ, which controls the number of iterations in the vertex pruning phase, and the number of iterations decreases with an increase of ϵ. From Figure 12(a) and Figure 12(b), we can observe that the time cost of SAC-RotC$^+$ and SAC-Exact$^+$ is almost the same when ϵ is very small because the cost of processing the vertex pruning phase dominates the cost in the second phase. On `Foursquare`, SAC-RotC$^+$ outperforms SAC-Exact$^+$ when ϵ is larger than 10^{-3}. Also, on `Gowalla`, SAC-RotC$^+$ is about one order of magnitude faster than SAC-Exact$^+$ when ϵ is larger than 10^{-4}. This is because our SAC-RotC$^+$ algorithm obtains the result incrementally and significantly outperforms the triple-vertex-based algorithm in the second phase, which incurs the dominating time cost as ϵ gets large. Comparing the two algorithms' minimal time cost on both datasets, we can conclude that SAC-RotC$^+$ can achieve a speed-up around twice.

5.4 Efficiency Evaluation of the algorithms to solve the diversified RB-k-core search problem

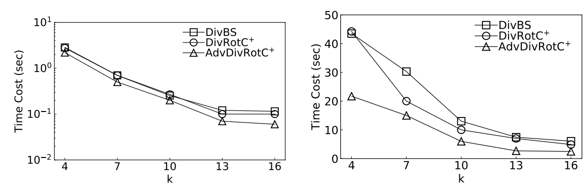
In this section, we evaluate the efficiency of the proposed algorithms (i.e., DivBS, DivRotC$^+$, and AdvDivRotC$^+$) to solve the diversified top-t RB-k-core search problem under different settings of parameters. Firstly, we evaluate the effect of t. Then, we evaluate the effect of k. We also evaluate the scalability (i.e., the effect of graph size and the effect of r) of these algorithms.

Evaluating the effect of t in diversified RB-k-core search. In Figure 13, we evaluate the effect of t for the three diversified algorithms on `Gowalla` and `Foursquare`. We vary t from 1 to 9 and fix the other parameters as default values. We can observe that when $t = 1$, the algorithms DivRotC$^+$ and AdvDivRotC$^+$ are much faster than DivBS on both datasets. This is because these two algorithms only need to maintain one RB-k-core in the result set, while DivBS needs to maintain all the RB-k-cores. When $t \geq 3$, the time cost of all the algorithms slightly increases when t increases. As expected, AdvDivRotC$^+$ outperforms DivBS and DivRotC$^+$ on both datasets because it only maintains top-t RB-k-cores with advanced pruning strategies.



(a) Gowalla, varying t (b) Foursquare, varying t

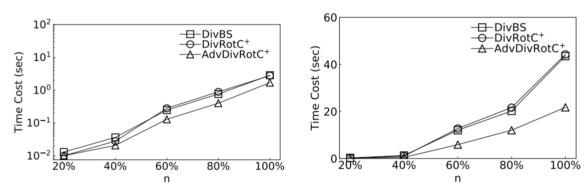
Figure 13: Diversified RB-k-core search - effect of t



(a) Gowalla, varying k (b) Foursquare, varying k

Figure 14: Diversified RB-k-core search - effect of k

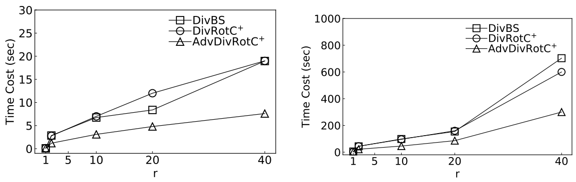
Evaluating the effect of k in diversified RB-k-core search. In Figure 14, we evaluate the effect of k for the algorithms DivBS, DivRotC$^+$ and AdvDivRotC$^+$ on `Gowalla` and `Foursquare`. We vary k from 4 to 16 and fix the other parameters as default values. We can observe that the time cost of all these algorithms drops when k increases. This is because the number of candidate vertices is reduced with an increase of k. Also, as an advanced solution, AdvDivRotC$^+$ outperforms DivRotC$^+$ and DivBS on both datasets.



(a) Gowalla, varying n (b) Foursquare, varying n

Figure 15: Diversified RB-k-core search - effect of graph size

Scalability. (1) *Evaluating the effect of graph size in diversified RB-k-core search.* Figure 15 shows the scalability of four algorithms by varying n from 20% to 100% in all datasets. We can observe that, on both datasets, our advanced algorithm AdvDivRotC$^+$ is scalable and its computation cost increases when the percentage of vertices increases. As discussed before, AdvDivRotC$^+$ is more efficient than the other two algorithms.



(a) Gowalla, varying r (b) Foursquare, varying r

Figure 16: Diversified RB-k-core search - effect of r

(2) *Evaluating the effect of r in diversified RB-k-core search.* Figure 16 shows the effect of r on `Gowalla` and `Foursquare`. We vary r from 1km to 40km and fix the other parameters as default values. In Figures 16(a) and 16(b), the time cost of these three algorithm increase as r increases and AdvDivRotC$^+$ is much faster than DivBS and DivRotC$^+$ when r is large. For instance, in Figure 16(b), AdvDivRotC$^+$ is more than three times faster than DivBS and DivRotC$^+$ when $r = 40$km.

6 Related Work

Community retrieval has been widely studied and used in many applications such as location-aware marketing [27], influence analysis [22], and event recommendation [24].

Community retrieval considering social connections. Prior works study various models such as k-core [21], [26], [32], k-truss [8], [19], [31], [35], [43], and clique [25] to retrieve communities based on users' social connections. Based on k-core, [5], [9], [33] study algorithms for k-core community search. Based on k-truss, Huang et al. [20] study the closest model and the triangle-connected model for community search are studied in [2], [18]. In [40], Yuan et al. propose algorithms to solve the densest clique percolation community search problem. However, the geo-locations of users are not considered in the above works.

Community retrieval considering spatial locations. In spatial databases, several works study the group objects retrieval problem based on users' spatial locations such as [16], [30], [38] and [11]. Guo et al. [16] study the spatial keyword query which retrieves a group of objects close to each other and cover a set of keywords together. Wu et al. [38] adapt the densest subgraph model to the spatial community search problem on dual networks. The work [30] proposes localitySeach which retrieves top-k sets of spatial web objects by integrating spatial distance, textual relevance, and a "co-locality" measure into one ranking function. The work [11] focuses on context-aware search over social media data. It analyses the data-centric challenges in temporal, spatial, and spatio-temporal contexts. These proposals do not consider the social connections of users, and thus they are different from our problem.

Community retrieval considering both social connections and spatial locations. On geo-social networks, recently, some works study the community retrieval problem [7], [14], [38], [42], [44] considering both the spatial and social features. The works [7], [42] mainly focus on analyzing and understanding the complexity networks rather than online community search. The most closely related work of radius-bounded k-core computation is that Zhu et al. study finding a community within a given rectangle in [44]. Their study is different from our work because what we consider is restricting the size of community spatially instead of within a given rectangle. Fang et al. [14] propose both exact and approximate algorithms to find a community covered by the smallest circle for a given query vertex. In their work, the radius of a circle is not given by users and only one community covered by the smallest circle is returned to users, and thus it cannot provide more options for users as done by our work.

Diversified top-t search. In the literature, many works [1], [3], [6], [10], [13], [23], [28], [29], [34], [39], [41] study to find diversified top-t answers according to a specific problem. In these works, Yuan et al. [39] aim to find diversified top-k cliques. [1], [3] studied diversified top-t document retrieval. Lin et al. [23] focus on the t most representative skyline problem. The diversified top-t graph pattern matching problem is studied in [13]. The diversified (k, r)-core search problem is studied in [41]. However, since the problems and models are different, none of them can be directly used to solve the diversified top-t RB-k-core search problem.

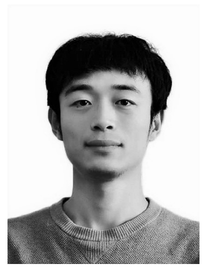
7 Conclusion

In this paper, we study the RB-k-core search problem. We propose a triple-vertex-based algorithm and a binary-vertex-based algorithm as benchmark algorithms to find all the RB-k-cores. We propose a rotating-circle-based algorithm which can find possible cost sharing opportunities. The rotating-circle-based algorithm is further enhanced by critical pruning techniques. In addition, we study the diversified RB-k-core search problem which aims to find representative RB-k-cores with rich information. We conduct extensive experiments on both real and synthetic datasets and the experimental result shows that our rotating-circle-based algorithm significantly outperforms the benchmark algorithms.

References

[1] R. Agrawal, S. Gollapudi, A. Halverson, and S. Ieong. Diversifying search results. In *Proceedings of the second ACM international conference on web search and data mining*, pages 5–14. ACM, 2009.
[2] E. Akbas and P. Zhao. Truss-based community search: a truss-equivalence based indexing approach. *PVLDB*, 10(11):1298–1309, 2017.
[3] A. Angel and N. Koudas. Efficient diversity-aware search. In *SIGMOD*, pages 781–792. ACM, 2011.
[4] G. Ausiello, N. Boria, A. Giannakos, G. Lucarelli, and V. T. Paschos. Online maximum k-coverage. *Discrete Applied Mathematics*, 160(13-14):1901–1913, 2012.
[5] N. Barbieri, F. Bonchi, E. Galimberti, and F. Gullo. Efficient and effective community search. *Data mining and knowledge discovery*, 29(5):1406–1433, 2015.
[6] A. Borodin, H. C. Lee, and Y. Ye. Max-sum diversification, monotone submodular functions and dynamic updates. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGAI symposium on Principles of Database Systems*, pages 155–166. ACM, 2012.
[7] Y. Chen, J. Xu, and M. Xu. Finding community structure in spatially constrained complex networks. *IJGIS*, 29(6):889–911, 2015.
[8] J. Cohen. Trusses: Cohesive subgraphs for social network analysis. *NSATR*, 16, 2008.
[9] W. Cui, Y. Xiao, H. Wang, and W. Wang. Local serach of communities in large graphs. In *SIGMOD*, pages 991–1002, 2014.
[10] T. Deng and W. Fan. On the complexity of query result diversification. *PVLDB*, 6(8):577–588, 2013.

[11] L. R. Derczynski, B. Yang, and C. S. Jensen. Towards context-aware search and analysis on social media data. In *Proceedings of the 16th international conference on extending database technology*, pages 137–142. ACM, 2013.
[12] J. Elzinga and D. W. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6(4):379–394, 1972.
[13] W. Fan, X. Wang, and Y. Wu. Diversified top-k graph pattern matching. *PVLDB*, 6(13):1510–1521, 2013.
[14] Y. Fang, R. Cheng, X. Li, S. Luo, and J. Hu. Effective community search over large spatial graphs. *PVLDB*, 10(6):709–720, 2017.
[15] U. Feige. A threshold of ln n for approximating set cover. *JACM*, 45(4):634–652, 1998.
[16] T. Guo, X. Cao, and G. Cong. Efficient algorithms for answering the m-closest keywords query. In *SIGMOD*, pages 405–418. ACM, 2015.
[17] R. Hartshorne. *Geometry: Euclid and beyond*. Springer Science & Business Media, 2013.
[18] X. Huang, H. Cheng, L. Qin, W. Tian, and J. X. Yu. Querying k-truss community in large and dynamic graphs. In *SIGMOD*, pages 1311–1322, 2014.
[19] X. Huang and L. V. Lakshmanan. Attribute-driven community search. *PVLDB*, 10(9):949–960, 2017.
[20] X. Huang, L. V. Lakshmanan, J. X. Yu, and H. Cheng. Approximate closest community search in networks. *PVLDB*, 9(4):276–287, 2015.
[21] W. Khaouid, M. Barsky, V. Srinivasan, and A. Thomo. K-core decomposition of large networks on a single pc. *PVLDB*, 9(1):13–23, 2015.
[22] M. Kitsak, L. K. Gallos, S. Havlin, F. Liljeros, L. Muchnik, H. E. Stanley, and H. A. Makse. Identification of influential spreaders in complex networks. *arXiv*, 2010.
[23] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting stars: The k most representative skyline operator. In *ICDE*, pages 86–95. IEEE, 2007.
[24] X. Liu, Q. He, Y. Tian, W.-C. Lee, J. McPherson, and J. Han. Event-based social networks: linking the online and offline social worlds. In *SIGKDD*, pages 1032–1040. ACM, 2012.
[25] R. D. Luce and A. D. Perry. A method of matrix analysis of group structure. *Psychometrika*, 14(2):95–116, 1949.
[26] D. W. Matula and L. L. Beck. Smallest-last ordering and clustering and graph coloring algorithms. *JACM*, 30(3):417–427, 1983.
[27] D. McKenzie-Mohr. *Fostering sustainable behavior: An introduction to community-based social marketing*. New society publishers, 2011.
[28] E. Minack, W. Siberski, and W. Nejdl. Incremental diversification for very large sets: a streaming-based approach. In *SIGIR*, pages 585–594. ACM, 2011.
[29] L. Qin, J. X. Yu, and L. Chang. Diversifying top-k results. *PVLDB*, 5(11):1124–1135, 2012.
[30] Q. Qu, S. Liu, B. Yang, and C. S. Jensen. Efficient top-k spatial locality search for co-located spatial web objects. In *MDM*, volume 1, pages 269–278. IEEE, 2014.
[31] K. Saito, T. Yamada, and K. Kazama. Extracting communities from complex networks by the k-dense method. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 91(11):3304–3311, 2008.
[32] S. B. Seidman. Network structure and minimum degree. *Social networks*, 5(3):269–287, 1983.
[33] M. Sozio and A. Gionis. The community-search problem and how to plan a succesful cocktail party. In *SIGKDD*, pages 939–948, 2010.
[34] M. R. Vieira, H. L. Razente, M. C. Barioni, M. Hadjieleftheriou, D. Srivastava, C. Traina, and V. J. Tsotras. On query result diversification. In *ICDE*, pages 1163–1174. IEEE, 2011.
[35] J. Wang and J. Cheng. Truss decomposition in massive networks. *PVLDB*, 5(9):812–823, 2012.
[36] K. Wang, X. Cao, X. Lin, W. Zhang, and L. Qin. Efficient computing of radius-bounded k-cores. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*, pages 233–244. IEEE, 2018.
[37] G. Watson. Computing helmert transformations. *JCAM*, 197(2):387–394, 2006.
[38] Y. Wu, R. Jin, X. Zhu, and X. Zhang. Finding dense and connected subgraphs in dual networks. In *ICDE*, pages 915–926. IEEE, 2015.
[39] L. Yuan, L. Qin, X. Lin, L. Chang, and W. Zhang. Diversified top-k clique search. *VLDBJ*, 25(2):171–196, 2016.
[40] L. Yuan, L. Qin, W. Zhang, L. Chang, and J. Yang. Index-based densest clique percolation community search in networks. *TKDE*, 30(5):922–935, 2017.
[41] F. Zhang, X. Lin, Y. Zhang, L. Qin, and W. Zhang. Efficient community discovery with user engagement and similarity. *VLDBJ*, pages 1–26, 2019.
[42] F. Zhang, Y. Zhang, L. Qin, W. Zhang, and X. Lin. When engagement meets similarity: efficient (k, r)-core computation on social networks. *PVLDB*, 10(10):998–1009, 2017.
[43] Y. Zhang and S. Parthasarathy. Extracting analyzing and visualizing triangle k-core motifs within networks. In *ICDE*, pages 1049–1060. IEEE, 2012.
[44] Q. Zhu, H. Hu, C. Xu, J. Xu, and W.-C. Lee. Geo-social group queries with minimum acquaintance constraints. *VLDBJ*, pages 1–19, 2014.

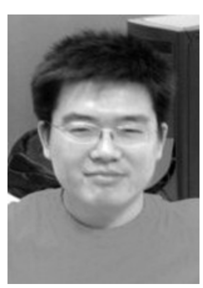


Kai Wang received the BEng degree in Computer Science from Zhejiang University in 2016, and the PhD degree in Computer Science from the University of New South Wales in 2020. He is currently a research associate in the School of Computer Science and Engineering, University of New South Wales. His research interests lie in big data analytics, especially for the graph/network and spatial data.

Shuting Wang is currently working at Zhejiang Gongshang University. She is also conducting research as a volunteer researcher at the East China Normal University in data science Lab. She received Bachelor degree in 2014 from Jiangxi Normal university and Research-Master degree in 2017 from Zhejiang Gongshang University.

Xin Cao is currently a senior lecturer in the School of Computer Science and Engineering (CSE) at The University of New South Wales (UNSW). His research interests include Databases, Data Mining, and Artificail Intelligence. He received the Ph.D. degree from Nanyang Technological University (NTU), Singapore, in 2014.

Wenjie Zhang received the PhD degree in computer science and engineering from the University of New South Wales, in 2010. She is currently an associate professor and ARC DECRA (Australian Research Council Discovery Early Career Researcher Award) fellow in the School of Computer Science and Engineering, the University of New South Wales, Australia. Her research interests lie in big data management and processing.



Lu Qin received the BEng degree from the Department of Computer Science and Technology, Renmin University of China, in 2006, and the PhD degree from the Department of Systems Engineering and Engineering Management, Chinese University of Hong Kong, in 2010. He is currently an associate professor with the Centre for Artificial Intelligence, University of Technology, Sydney. His research interests include big graph analytics and graph query processing.