Efficient Algorithms for Answering the m-Closest Keywords Query

Tao Guo[†] Xin Cao[‡] Gao Cong[†]
[†]Nanyang Technological University, Singapore
[†]tguo001@e.ntu.edu.sg, [†]gaocong@ntu.edu.sg
[‡]Queen's University Belfast, United Kingdom
[‡]x.cao@qub.ac.uk

ABSTRACT

As an important type of spatial keyword query, the m-closest keywords (mCK) query finds a group of objects such that they cover all query keywords and have the smallest diameter, which is defined as the largest distance between any pair of objects in the group. The query is useful in many applications such as detecting locations of web resources. However, the existing work does not study the intractability of this problem and only provides exact algorithms, which are computationally expensive.

In this paper, we prove that the problem of answering mCK queries is NP-hard. We first devise a greedy algorithm that has an approximation ratio of 2. Then, we observe that an mCK query can be approximately answered by finding the circle with the smallest diameter that encloses a group of objects together covering all query keywords. We prove that the group enclosed in the circle can answer the mCK query with an approximation ratio of $\frac{2}{\sqrt{3}}$. Based on this, we develop an algorithm for finding such a circle exactly, which has a high time complexity. To improve efficiency, we propose another two algorithms that find such a circle approximately, with a ratio of $(\frac{2}{\sqrt{3}}+\epsilon)$. Finally, we propose an exact algorithm that utilizes the group found by the $(\frac{2}{\sqrt{3}}+\epsilon)$-approximation algorithm to obtain the optimal group. We conduct extensive experiments using real-life datasets. The experimental results offer insights into both efficiency and accuracy of the proposed approximation algorithms, and the results also demonstrate that our exact algorithm outperforms the best known algorithm by an order of magnitude.

Categories and Subject Descriptors

H.2.8 [**Database Management**]: Database Applications—*Spatial databases and GIS*

Keywords

Spatial keyword query; Geo-textual objects

1. INTRODUCTION

SIGMOD'15, May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2723723.

With the proliferation of GPS-equipped mobile devices, massive amounts of geo-textual objects are becoming available on the web that each possess both a geographical location and a textual description. For example, such geo-textual objects include points of interest (POIs) associated with texts, such as tourist attractions, hotels, restaurants, businesses, entertainment services, etc. Other example geo-textual objects include geo-tagged micro-blogs (e.g., Tweets), photos with both tags and geo-locations in social photo sharing websites (e.g., Flickr), and check-in information on places in location-based social networks (e.g., FourSquare).

The availability of substantial amount of geo-textual objects gives prominence to the spatial keyword queries that target these objects, which have been studied extensively in recent years [3, 5, 7, 9, 20, 21, 23]. Typically, a spatial keyword query finds the objects that best match the arguments in the query exploiting both locations and textual descriptions. Such queries are widely used in many services and applications, such as online Map services, travel itinerary planning, etc.

As an important type of the spatial keyword query, the m-closest keywords (mCK) query [21, 22] is defined for finding a set of closest keywords in the geo-textual object database. Specifically, let $\mathcal{O}$ be a set of geo-textual objects, and each object $o \in \mathcal{O}$ has a location denoted by $o.\lambda$ and a textual description $o.\psi$. The mCK query q contains m keywords, and it finds a group of objects G such that they cover all the query keywords (i.e., $q \subseteq \bigcup_{o \in G} o.\psi$) and such that the diameter of this group, denoted by $\delta(G)$, is minimized. The diameter of a group is defined as the maximum distance between any pair of objects in the group.

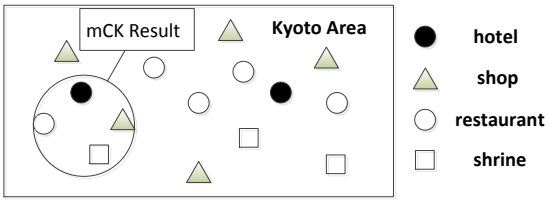


Figure 1: Example of the mCK query

The mCK query has many applications as shown in the proposals [21, 22]. For example, it can be used in detecting geographic locations of web resources such as documents or photos. Given a document or a photo with some tags, we can issue an mCK query using these tags. After a group of objects covering all tags that have the smallest diameter is found, the area where these objects locate is very likely to be the location of the document or the photo. It has been shown [21, 22] that this approach can address the challenge faced by the traditional location detection techniques when the tags of documents or photos do not contain gazetteer terms.

The mCK query also has potential applications for location-based service providers [22]. One example application is "fans of Apple products can submit 'Apple store subway' to locate a retailer store near the subway for convenient purchase of the products in New York [22]." As another example, consider a tourist who is planning for a trip to Kyoto. She wishes to explore an area where the following attractions are within walking distance (i.e., close to one another): $shrine$, $shop$, $restaurant$, and $hotel$. That is, she is seeking a location to stay and do sightseeing and shopping on foot. This can be formulated as an mCK query. As shown in Figure 1, we can perform search on objects within the area of Kyoto, and the group enclosed in the circle is returned to meet the tourist's requirements.

Exact algorithms are proposed in the studies [21,22] that run exponentially with the number of objects relevant to the query, and thus they are computationally prohibitive when the number of relevant objects is large. For example, in our experiments, the best known algorithm [22] took almost 1 hour to answer a query containing 8 keywords on a dataset with 1 million objects. Moreover, the hardness of the problem is still unknown.

In this paper, we establish that the problem of exactly answering the mCK query is NP-hard, which can be proven by a reduction from the 3-SAT problem. The intractability result motivates us to design approximation algorithms for efficiently processing the mCK query. We first develop a greedy approach that has an approximation ratio of 2. We call this algorithm the Greedy Keywords Group algorithm, denoted by *GKG*. Utilizing the result returned by *GKG*, we propose three non-trivial approximation algorithms that all have better performance guarantee than *GKG*. Based on one of them, we further develop an efficient exact algorithm.

The three approximation algorithms answer the mCK query by finding the circle with the smallest diameter that encloses a group of objects together covering all query keywords. We call such a circle the "smallest keywords enclosing circle," and given a query q, we denote the circle by SKEC_q. We prove that the group in SKEC_q can answer the mCK query with an approximation ratio of $\frac{2}{\sqrt{3}}$. Although finding SKEC_q is solvable in polynomial-time, it is still an open problem to find SKEC_q efficiently, which is challenging because we know neither its radius nor its center.

We first develop an approach to finding SKEC_q exactly. We denote the set of objects that contain at least one query keyword by $\mathcal{O}'$. This algorithm is based on a lemma we establish: there must exist either three or two objects in $\mathcal{O}'$ on the boundary of SKEC_q and they determine SKEC_q. Unfortunately, this method has a high time complexity ($|\mathcal{O}'|^4$ in the worst case). We call this method the Smallest Keywords Enclosing Circle (denoted by *SKEC*, with a bit abuse of notation). This method is impractical when $|\mathcal{O}'|$ is large.

For better efficiency, we propose to find SKEC_q approximately. First, we develop an algorithm that performs search on each object o in $\mathcal{O}'$ one by one. We call this method the Approximate Smallest Keywords Enclosing Circle (denoted by *SKECa*) algorithm. We prove that the group enclosed in the circle found by *SKECa* can answer the mCK query with an approximation ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$ (ϵ is an arbitrarily small positive value). To further improve efficiency, we devise techniques to perform the search on all objects in $\mathcal{O}'$ together, instead of on each of them separately. This algorithm finds the same circle as found by *SKECa*. We denote this enhanced algorithm by *SKECa+*. Both algorithms have better time complexity than that of *SKEC*.

Finally, based on *SKECa+*, we devise an exact algorithm for the mCK query. Since answering mCK queries is NP-hard, it is challenging to devise an efficient exact algorithm—an exhaustive search on the object space cannot be avoided. We prove that the diameter of the circle that encloses the optimal group cannot exceed $\frac{2}{\sqrt{3}}$ times the diameter of the group found in *SKECa+*. Thus, we do an exhaustive search around each object o in $\mathcal{O}'$, considering only objects in $\mathcal{O}'$ whose distances to o are smaller than $\frac{2}{\sqrt{3}}$ times the diameter of group G returned by *SKECa+*. We denote this method by *EXACT*. The group G found by *SKECa+* is able to greatly reduce the search space of *EXACT*.

In summary, the contributions of this paper are twofold. First, we prove that the problem of answering mCK queries is NP-hard. We observe and prove that an mCK query q can be answered by SKEC_q approximately with a performance guarantee. Based on this, we present novel approximation algorithms, *SKEC*, *SKECa*, and *SKECa+*, all with provable performance bounds for answering the mCK query. We also design an exact approach based on *SKECa+* with the worst case time complexity $O(|\mathcal{O}'|n^{|q|-1})$, where n is the number of objects in a region around an object, which can be bounded by the result of *SKECa+*, and in general $n \ll \mathcal{O}'$. Its complexity is better than that of the best known algorithm [22], which is $O(|\mathcal{O}'|^{|q|})$. Second, we conduct extensive experiments on real-life datasets. The experimental results demonstrate that the proposed approximation algorithms offer scalability and excellent efficiency and accuracy and that the exact algorithm outperforms the best known algorithm for answering mCK [22] by orders of magnitude.

2. PROBLEM AND BACKGROUND

2.1 Problem Statement

Let $\mathcal{O}$ be a database consisting of a set of geo-textual objects. Each object $o \in \mathcal{O}$ is associated with a location $o.\lambda$ and a set of keywords $o.\psi$ describing the object (e.g., the menu of restaurants).

Definition 1: Diameter of a group: Given a group of objects G, its diameter is defined as the maximum Euclidean distance between any pair of objects in G, denoted by $\delta(G)$. That is, $\delta(G) = \max_{o_i,o_j \in G} \mathsf{Dist}(o_i, o_j)$, where $\mathsf{Dist}(o_i, o_j)$ computes the Euclidean distance between o_i and o_j.

Definition 2: Problem definition [21,22]: An *m-closest keywords* (mCK) query q contains m keywords $\{t_{q_1}, t_{q_2}, ..., t_{q_m}\}$, and it finds a group of objects $G \subseteq \mathcal{O}$, each containing at least one query keyword, such that $\cup_{o \in G} o.\psi \supseteq q$ and such that $\delta(G)$ is minimized.

Definition 3: Feasible group: Given an mCK query q, if a group of objects can cover all keywords in q, we call such a group a "feasible group" or a "feasible solution".

We establish the hardness of answering the mCK query by the following theorem.

Theorem 1: The problem of answering mCK queries is NP-hard.

PROOF. See Appendix A $\square$

2.2 Existing Solutions for mCK Queries

bR*-tree Based Method: In the work [21], Zhang et al. propose a hybrid index structure that combines the R*-tree and bitmap, named bR*-tree. Each R*-tree node is augmented with a bitmap indicating the keywords contained in the objects rooted at this node. Based on the bR*-tree, an exact method is proposed to answer the mCK query. The algorithm adopts an exhaustive search enhanced with several pruning strategies. The search starts from the root node, and is performed in a top-down manner. In each level of the tree, all the candidate combinations of nodes (each combination can cover all query keywords) are generated. For each such combination of nodes, all combinations of their child nodes that cover all query keywords are generated. This process is repeated

until the leaf node level is reached, and then all possible groups are enumerated and the best one is returned as the result.

Virtual bR*-tree Based Method: In the subsequent work [22], an improved version of the bR*-tree called virtual bR*-tree is proposed. During query processing, the relevant objects and R*-tree nodes are read from the inverted file, and a virtual bR*-tree is built using these relevant objects and nodes in a bottom-up way. The query is then processed using the algorithm proposed in the work [21] based on the virtual bR*-tree. Compared to the original bR*-tree, the size of the tree is significantly reduced. The experimental results show that this solution is much more efficient than that proposed in the earlier work [21]. Hence, we use the virtual bR*-tree based method as the baseline, denoted by *VirbR*.

Spatial group keyword query: Cao et al. [2] study the spatial group keyword (SGK) query. Such a query Q has both a location $Q.\lambda$ and a set of query keywords $Q.\psi$. Given a database of geo-textual objects $\mathcal{O}$, it retrieves a set of objects G, such that $\cup_{o\in G} o.\psi \supseteq Q.\psi$, and the cost of G w.r.t. Q is minimized. The cost function takes into account both the distance of the group to the query and the inter-object distance. That is, the returned group is close to the query location, and the group also has a small diameter. When considering only the diameter of the group and ignoring the query location, such a query is equivalent to the mCK query. The pruning methods using location $Q.\lambda$ become invalid in the algorithm proposed for the SGK query, which is reduced to the bR*-tree based algorithm [21]. Thus it is not compared in our experiments.

Long et al. [16] propose algorithms for processing the SGK query, but the algorithms cannot handle the case when considering only the inter-object distance, and thus they cannot be used to answer the mCK query. They also studied a variant of the SGK query, where the cost function of a group G is defined as $\max_{o_1,o_2\in\{G\cup Q\}}(\mathsf{Dist}(o_1,o_2))$, called Dia-CoSKQ. That is, the query location is also considered when computing the diameter of the group. They propose both exact and approximation algorithms for this query. We adapt the two algorithms to answer the mCK query. Give an mCK query q, we select the most infrequent keyword t_{inf}, and on each object o_i containing t_{inf}, we issue a Dia-CoSKQ query with o_i as the query location and $q \setminus o_i.\psi$ as the query keywords, and we invoke the algorithm (either exact or approximate) [16] to answer the query. After all objects containing t_{inf} are processed, the group with the best cost is used as the result of the mCK query. The reason for choosing the most infrequent keyword t_{inf} is to minimize the number of times for issuing the Dia-CoSKQ query. The two algorithms are denoted by *ASGK* (adapted SGK exact) and *ASGKa* (adapted SGK approximation), respectively. We compare *ASGK* and *ASGKa* with our proposed algorithms in the experiments, and they both have poor performance. The result shows that the adaptation is not suitable for processing the mCK query.

3. ALGORITHM *GKG*

We develop a 2-approximation algorithm as a baseline for answering the mCK query. We call it the Greedy Keyword Group (*GKG*) algorithm. The algorithm is described as follows. Given a query $q = \{t_{q_1}, t_{q_2}, ..., t_{q_m}\}$, we first find the most infrequent keyword t_{inf} among the keywords in q based on their frequencies in dataset $\mathcal{O}$. Then, around each object o containing t_{inf}, for each keyword $t \in q \setminus o.\psi$ we find the nearest object containing t. These objects and o form a feasible group, and we denote this group by G_o. After all the objects containing t_{inf} are processed, we select the group that has the smallest diameter to answer the query approximately. We find the most infrequent keyword t_{inf} because this can reduce the number of subsequent operations for finding the nearest object. In the algorithm, we utilize the virtual bR*-tree indexing structure [22] to find the nearest object containing a term t. The reason for using the virtual bR*-tree is that we use the same index for all methods as the *VirbR* algorithm [22] for a fair comparison of different algorithms. Alternatively, we can also use other geo-textual indexes (such as IR-tree [7]). The algorithm details are presented in Appendix B.

Approximation ratio and Complexity. We proceed to prove that *GKG* is within an approximation factor of 2. We denote the group returned by *GKG* by G_{gkg}, and denote the optimal group for the query by G_{opt}.

Theorem 2: $\delta(G_{gkg}) \leq 2 \cdot \delta(G_{opt})$.

PROOF. Let o_i to be the object containing t_{inf} in G_{opt}, and we denote the feasible group containing o_i by G_{o_i}. We know that $\delta(G_{gkg}) \leq \delta(G_{o_i})$ according to the algorithm. We denote by o_f the object that is the furthest to o_i in G_{o_i}.

On one hand, because all objects in G_{o_i} fall in the circle with o_i as center and $\mathsf{Dist}(o_i, o_f)$ as radius, $\delta(G_{o_i})$ must be no larger than the diameter of the circle, i.e., $2\mathsf{Dist}(o_i, o_f)$. On the other hand, o_f must cover a keyword t_f that is not covered by other objects in G_{o_i}, and it is the nearest object to o_i containing t_f. Hence, in G_{opt}, the object containing t_f must be no closer to o_i than o_f. We thus know that $\delta(G_{opt}) \geq \mathsf{Dist}(o_i, o_f)$.

Hence, $\delta(G_{gkg}) \leq \delta(G_{o_i}) \leq 2\mathsf{Dist}(o_i, o_f) \leq 2\delta(G_{opt})$. □

In *GKG*, each object containing t_{inf} is considered to be in the candidate group, and we denote the set of such objects as $\mathcal{O}_{t_{inf}}$. For each object $o \in \mathcal{O}_{t_{inf}}$, we find at most $m-1$ other objects together with o to form a feasible group. If we assume finding the nearest object containing a given keyword costs time d, the time complexity of *GKG* is $O(m|\mathcal{O}_{t_{inf}}|d)$, where d depends on the index structure in place.

4. SKEC-BASED ALGORITHMS

To achieve better accuracy, we propose several novel algorithms with much smaller approximation ratios, which answer the mCK query by finding a circle with the smallest diameter that encloses a group of objects covering all query keywords. We call such a circle the "smallest keywords enclosing circle" w.r.t. the given query q, denoted by SKEC_q. We prove that the group enclosed in the circle SKEC_q can answer q with a ratio of $\frac{2}{\sqrt{3}}$, which is very close to 1.

However, finding SKEC_q efficiently remains an open problem, which is challenging because neither its radius nor its center is known, although it is solvable in polynomial-time. We first develop an algorithm for finding SKEC_q exactly and it can answer mCK query q with a ratio of $\frac{2}{\sqrt{3}}$. We denote this method by *SKEC*. Algorithm *SKEC* has a high time complexity. To achieve better efficiency, we propose to find SKEC_q approximately, and design two algorithms *SKECa* and *SKECa+*, both of which are able to answer the mCK query with a ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$, where ϵ is an arbitrarily small positive value.

We next introduce several definitions and theorems in Section 4.1, which lay the foundation of our algorithms. We detail *SKEC*, *SKECa*, and *SKECa+* in Sections 4.2, 4.3, and 4.4, respectively.

4.1 Minimum Covering Circle and Keywords Enclosing Circle

Definition 4: Minimum Covering Circle. Given a set of geo-textual objects G, the *Minimum Covering Circle* of G is the circle that encloses them with the smallest diameter, denoted by MCC_G.

The problem of finding the minimum covering circle for a given set of objects has been well studied [10,11,17]. Note that the diameter of the circle that encloses a group is different from the diameter

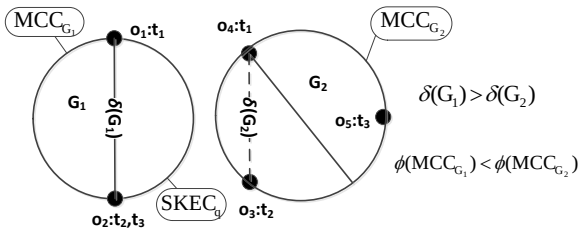


Figure 2: Example of two diameters

of the group. We denote the diameter of a circle $\mathcal{C}$ by $\phi(\mathcal{C})$, and we denote the diameter of a group G by $\delta(G)$. Given a group of objects G, we show that $\delta(G) \neq \phi(\mathsf{MCC}_G)$ in Figure 2. Both groups G_1 and G_2 cover keywords $\{t_1, t_2, t_3\}$. In group G_1, the two diameters are the same. However, in group G_2, the diameter of G_2 and the diameter of the minimum covering circle of G_2 are different.

Although the diameter of the circle that encloses a group and the diameter of the group can be different, we have the following theorems to show their relationship.

Theorem 3: [11] Given a set of objects G, its smallest object enclosing circle can be determined by at most three points in G which lie on the boundary of the circle. If it is determined by only two points, then the line segment connecting those two points must be a diameter of the circle. If it is determined by three points, then the triangle consisting of those three points is not obtuse.

Theorem 4: Given a set of objects G and its minimum covering circle MCC_G, we have $\frac{\sqrt{3}}{2}\phi(\mathsf{MCC}_G) \leq \delta(G) \leq \phi(\mathsf{MCC}_G)$.

PROOF. See Appendix C □

Definition 5: Keywords Enclosing Circle. Given a database of geo-textual objects $\mathcal{O}$ and a set of keywords ψ, the *Keywords Enclosing Circle* w.r.t. ψ (denoted by KEC_ψ) is a circle that encloses a group of objects covering all the given keywords in ψ. We call the one with the smallest diameter the *Smallest Keywords Enclosing Circle* (denoted by SKEC_ψ)

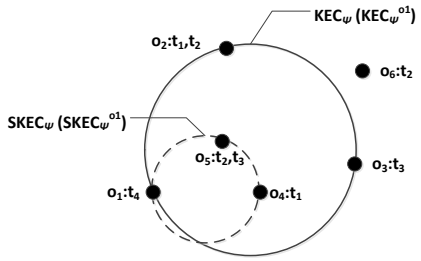


Figure 3: Examples of enclosing circles w.r.t. $\{t_1, t_2, t_3, t_4\}$.

Example 1: As shown in Figure 3, given a set of keywords $q = \{t_1, t_2, t_3, t_4\}$, the larger circle is a keyword enclosing circle, and the smaller circle with dashed line is the smallest keyword enclosing circle w.r.t. q.

From Figure 2, it can also be observed that given an mCK query q, the group enclosed by SKEC_q is not necessary the optimal group of q, i.e., G_{opt}. Consider the objects as shown in Figure 2, given a query $q = \{t_1, t_2, t_3\}$, G_2 is the optimal group rather than G_1 because $\delta(G_2) < \delta(G_1)$. However, the minimum covering circle of G_1 is SKEC_q, because $\phi(\mathsf{MCC}_{G_2}) > \phi(\mathsf{MCC}_{G_1})$.

Fortunately, we can establish the relationship between the diameter of G_{opt} and the diameter of the group enclosed in SKEC_q, denoted by G_{skec} (i.e., SKEC_q is $\mathsf{MCC}_{G_{skec}}$), as follows.

Theorem 5: $\delta(G_{skec}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$.

PROOF. According to Theorem 4, $\phi(\mathsf{MCC}_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$, and $\delta(G_{skec}) \leq \phi(\mathsf{SKEC}_q)$. Since SKEC_q has the smallest diameter, we can obtain $\delta(G_{skec}) \leq \phi(\mathsf{SKEC}_q) \leq \phi(\mathsf{MCC}_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$. □

Theorem 5 shows that if we can find the smallest keywords enclosing circle SKEC_q for a given mCK query q, the group of objects G_{skec} enclosed in SKEC_q can approximately answer query q with a ratio of $2/\sqrt{3}$ (≈ 1.1547). This lays the foundation of our proposed algorithms that find SKEC_q to approximately answer query q, to be presented in the next three subsections.

4.2 Algorithm *SKEC*

The *SKEC* algorithm finds the smallest keywords enclosing circle SKEC_q exactly for a given query q. The algorithm has a high time complexity (to be analyzed later). It is based on the following corollary, which follows Theorem 3 and the definition of SKEC_q.

Corollary 1: There must exist either three or two objects on the boundary of SKEC_q, which determine the circle SKEC_q. If two objects determine SKEC_q, the line segment connecting them is the diameter of SKEC_q.

We denote by $\mathcal{O}'$ the set of objects that contain at least one query keyword. According to the corollary, we can check every combination of two and three objects in $\mathcal{O}'$ to find whether the circle determined by the combination encloses a group covering all query keywords and has the smallest diameter. The circle found finally must be SKEC_q. We introduce several pruning strategies to reduce the number of checking in *SKEC*. Before presenting the details of *SKEC*, we introduce the following definition.

Definition 6: Object-across Keywords Enclosing Circle. Given a set of keywords q and an object o, we call a keywords enclosing circle passing through o the *o-across Keywords Enclosing Circle* (denoted by KEC_q^o). The o-across keywords enclosing circle with the smallest diameter is called the *o-across Smallest Keywords Enclosing Circle* (denoted by SKEC_q^o).

Example 2: As shown in Figure 3, the larger circle is an o_1-across keywords enclosing circle, and the smaller circle is the o_1-across smallest keywords enclosing circle w.r.t. $q = \{t_1, t_2, t_3, t_4\}$.

Given a query q, there must exist an object containing at least one query keyword on the boundary of SKEC_q. Hence, if we find the object-across smallest keywords enclosing circle (SKEC_q^o) on each object o in $\mathcal{O}'$ (these objects are obtained using the virtual bR*-tree index), the one with the smallest diameter must be SKEC_q. That is, $\mathsf{SKEC}_q = \min_{o \in \mathcal{O}'} \mathsf{SKEC}_q^o$.

The algorithm is shown in Algorithm 1. We first obtain a group G_{gkg} using the *GKG* algorithm, and the diameter of its minimum covering circle $\mathsf{MCC}_{G_{gkg}}$ serves as the initial upper bound of the diameter of SKEC_q. We denote the current best checked circle by C_{cur}, and C_{cur} is initialized as $\mathsf{MCC}_{G_{gkg}}$ (lines 1–2). For each object o in $\mathcal{O}'$, if it covers all query keywords, we return this object (line 5). Otherwise, we find the smallest o-across keyword enclosing circle, and update C_{cur} if it has a smaller diameter (line 6). Finally, we return the objects in C_{cur} to answer q (lines 7–8).

Procedure findOSKEC(). We next present how to find SKEC_q^o around each object o in $\mathcal{O}'$. For finding SKEC_q^o, the search space comprises only objects whose distance to o is smaller than $\phi(C_{cur})$. This is because for any object o_f with $\mathsf{Dist}(o, o_f) \geq \phi(C_{cur})$, the circle passing by o_f and o must have a diameter no smaller than $\mathsf{Dist}(o, o_f)$, and thus is worse than C_{cur}. If these objects together cannot cover all query keywords, o does not need to be processed.

Recall that SKEC_q is determined by either three or two objects in $\mathcal{O}'$. We aim at searching whether the circle determined by object

Algorithm 1: *SKEC* (q)

```
1 G_gkg ← GKG (q) ;                                   // invoke GKG
2 C_cur ← MCC_{G_gkg};
3 O' ← objects containing at least one keyword in q;
4 foreach object o ∈ O' do
5 |   if o.ψ = q then return {o} ;
6 |   C_cur ←findOSKEC(o, C_cur);
7 G_skec ← objects in C_cur;
8 return G_skec;
```

o together with a second object o_j, and the circle determined by o, o_j and a third object o_m is SKEC_q. For each object o_j in the search space of o, we first consider the case when SKEC_q is determined by two objects. We check all the objects in the circle determined by the pair, and update C_{cur} if the circle covers all query keywords. We next consider the case when SKEC_q is determined by three objects. For each object o_m in the search space of o, we check if the circle determined by o, o_j, and o_m can cover all query keywords and has the smallest diameter among all checked circles. The diameter of the circle can be computed using the laws of sins and cosines.

The second object o_j is processed in ascending order of their distances to o. This assures that when we reach an object o_j with distance to o larger than $\emptyset(C_{cur})$, we can terminate the search around o immediately, because a circle passing by any further object and o must have a diameter larger than $\emptyset(C_{cur})$. After o and o_j are fixed, an object o_m is considered as the third object only if $\mathsf{Dist}(o_m, o) < \mathsf{Dist}(o_j, o)$ and $\mathsf{Dist}(o_m, o_j) < \emptyset(C_{cur})$. The first constraint is because further objects will be processed in subsequent steps, where they are used as the second object. The second constraint is because further objects, together with o and o_j, cannot determine a better circle than C_{cur}.

After all objects in $\mathcal{O}'$ are processed, it is assured that all object-across keywords enclosing circles are found. We use the best one to answer the given mCK query. The pseudo code is given in Procedure findOSKEC in Appendix D.

Approximation ratio and Complexity. Algorithm *SKEC* finds SKEC_q exactly and has an approximation ratio of $2/\sqrt{3}$ according to Theorem 5. *SKEC* utilizes the current best circle to prune the search space when finding SKEC_q^o. Assuming that there are n objects in the search space around an object in the worst case, the number of checks in procedure findOSKEC() is O(n^2). We also need to read the objects within a circle, which costs at most O(n). Hence, the time complexity is O($|\mathcal{O}'|n^3$). The high time complexity makes *SKEC* impractical especially when n is large (in the worst case $n = |\mathcal{O}'|$).

Considering the high complexity of *SKEC* for finding SKEC_q exactly, we next propose two algorithms for finding SKEC_q approximately with much better efficiency.

4.3 Algorithm *SKECa*

In *SKEC*, on each relevant object o we find the o-across smallest keywords enclosing circle (SKEC_q^o) exactly in procedure findOSKEC(). In the *SKECa* algorithm we propose to find SKEC_q^o approximately to gain better efficiency.

4.3.1 Finding SKEC_q^o Approximately

The idea of this algorithm is based on the following property of keywords enclosing circles.

Property 1: Given a set of keywords ψ and an object o, if there exists no o-across keywords enclosing circle (KEC_q^o) with diameter D, then no KEC_q^o exists whose diameter is smaller than D.

PROOF. This can be proven by contradiction. If there exists an o-across keywords enclosing circle $\mathcal{C}$ with diameter D' smaller than D, at object o we can draw a circumscribed circle $\mathcal{C}_{circ}$ of $\mathcal{C}$ with diameter D, and it is obvious that $\mathcal{C}_{circ}$ is also an o-across keywords enclosing circle, leading to a contradiction. □

The property inspires us to use binary search to find the diameter and the position of SKEC_q^o. Given a value D, if we can find a KEC_q^o on object o with diameter D, we know that D must be an upper bound of $\emptyset(\mathsf{SKEC}_q^o)$, and we will try smaller D in the subsequent search. Otherwise, if using value D we can find no KEC_q^o on object o, D must be a lower bound of $\emptyset(\mathsf{SKEC}_q^o)$ (Property 1), and we need to enlarge D in the subsequent search. We repeat this process until the gap between the upper bound and the lower bound is smaller than a certain threshold α.

Parameter α is the error tolerance of the binary search, and we present how to set this parameter smartly based on the result of algorithm *GKG* in Section 4.3.3. As to be shown in Section 4.3.3, by setting a query-dependent value for α based on the result of algorithm *GKG*, we are able to guarantee the approximation ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$ for *SKECa*, where ϵ is an arbitrarily small value.

We set the initial upper bound of $\emptyset(\mathsf{SKEC}_q^o)$ by the diameter of the current best circle, because we aim to find a circle with smaller diameter than the current one. The lower bound can be simply set to 0. However, in order to accelerate the binary search, we use the following lemma to improve the lower bound of $\emptyset(\mathsf{SKEC}_q^o)$.

Lemma 1: $\emptyset(\mathsf{SKEC}_q^o) \geq \delta(G_{gkg})/2$, where G_{gkg} is the group found by algorithm *GKG*.

PROOF. We denote the group enclosed in SKEC_q^o by G_q. We obtain $\emptyset(\mathsf{SKEC}_q^o) \geq \delta(G_q)$ (Theorem 4) $\geq \delta(G_{opt}) \geq \delta(G_{gkg})/2$ (Theorem 2). □

Procedure findAppOSKEC(o, $\delta(G_{gkg})$, C_{cur}, α)

```
1  searchUB ← ø(C_cur);
2  oskec ← circleScan(o, searchUB);
3  if oskec is null then return null ;
4  searchLB ← δ(G_gkg)/2;
5  while searchUB − searchLB > α do
6  |   diam ← (searchUB + searchLB)/2;
7  |   C ← circleScan(o, diam);
8  |   if C ≠ ∅ then
9  |   |   searchUB ← diam;
10 |   |   oskec ← C;
11 |   else searchLB ← diam ;
12 return oskec;
```

As described in Procedure findAppOSKEC(), we first set the upper bound of $\emptyset(\mathsf{SKEC}_q^o)$ by the diameter of the current best circle (line 1). Then we use $searchUB$ to search for an o-across keywords enclosing circle by invoking the function circleScan() (to be described in Section 4.3.2) (line 2). If no such circle exists, we do not need to search on o because $\emptyset(\mathsf{SKEC}_q^o)$ must be larger than the diameter of the current best circle according to Property 1 (line 3). We set the lower bound of $\emptyset(\mathsf{SKEC}_q^o)$ according to Lemma 1 (line 4). The binary search stops when the gap between $searchUB$ and $searchLB$ is smaller than α (lines 5–11).

By replacing the procedure findOSKEC() with findAppOSKEC() in Algorithm 1, we obtain the *SKECa* algorithm. *SKECa* finds SKEC_q approximately, and the diameter of the circle found is smaller than $\emptyset(\mathsf{SKEC}_q) + \alpha$.

4.3.2 Procedure circleScan()

In each step of the binary search in the *SKECa* algorithm, we invoke Procedure circleScan() to check if there exists a KEC_q^o with

diameter D on a given object o. This procedure is a key operation of Algorithm *SKECa*, and is non-trivial.

The idea of finding a KEC_q^o with diameter D is as follows: If there exists such a circle, besides o there must exist another object falling on the boundary of this circle (we can always rotate the circle around o to assure this). Hence, from the objects whose distance to o is smaller than D, we randomly select one and make it on the boundary of the circle, and we can fix the position of this circle. Then, we rotate this circle from this position, and if at a certain position the objects inside this circle can cover all the query keywords, we know that a KEC_q^o with diameter D is found. If we have rotated the circle back to the beginning position but still cannot find a feasible group, we know that no KEC_q^o with diameter D exists. Figure 4 illustrates the sweeping area around an object o given D as the diameter, which is the circle with dashed lines.

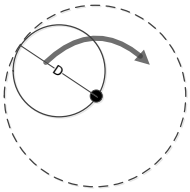


Figure 4: Sweeping area

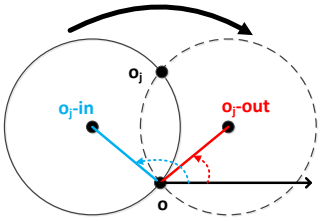


Figure 5: Computing two angles

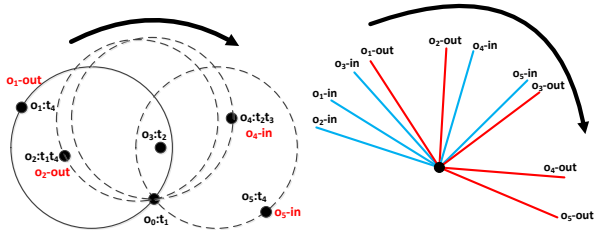
We use the virtual bR*-tree to read all objects in the sweeping area. Before the checking on an object o by rotating the circle, we first check whether all keywords can be covered by the objects in the sweeping area, and if not, there exists no KEC_q^o with diameter D and the checking on o is thus avoided.

Now we present how to efficiently rotate the circle to find a KEC_q^o with diameter D. We rotate the circle clockwise. Note that during the rotation, each object in the sweeping area falls on the boundary of the circle only twice. That is, each object is scanned outside-in or inside-out by the circle exactly once. We map the objects in the sweeping area around o to a polar coordinate system with o as the pole and the horizontal line pointing to the right as the polar axis. On each object o_j in the sweeping area, we compute two polar angles when o_j touches the boundary: 1) when o_j is entering the circle, we find the center point of the circle at the current position and we compute the polar angle of the center point, denoted by o_j-in; and 2) when o_j is exiting the circle, we compute the polar angle of the center point of the circle at the current position, denoted by o_j-out. Figure 5 shows an example of computing o_j-in and o_j-out in the polar coordinate system with o as the pole.

Next, after we compute the two polar angles for each object in the sweeping area, we sort all these angles in descending order and store them in $\angle$. Initially, we place the circle such that the center point of the circle has the polar angle equal to the largest angle in $\angle$. We check if the objects in this circle can cover all query keywords. If so, a KEC_q^o is found and we return the circle, and otherwise we record the keywords covered by these objects with their frequencies in a table Tab and we start the clockwise rotation from the current position. When the circle is rotated to a position such that the center point of the circle has a polar angle equal to the next largest angle in $\angle$, we update table Tab because the objects enclosed in the circle change. If the angle is an object inside-out angle, we remove from Tab the keywords covered by the object to be rotated out of the circle. If the angle is an object outside-in angle, we update Tab by adding in the keywords covered by the object to be rotated in the circle. If Tab contains all query keywords we terminate the checking, and otherwise we repeat the above rotation process until we find a KEC_q^o or all angles in $\angle$ are reached. The pseudo code is given in Procedure circleScan.

Procedure circleScan(o, $diam$)

```
1  ∠ ← empty List;
2  Initialize Tab by q;
3  G ← ∅;
4  foreach object o_j ∈ O' do
5      o_j-in ← getInAngle(o, diam, o_j);
6      o_j-out ← getOutAngle(o, diam, o_j);
7      ∠.addTuple(o_j-out, out, o_j);
8      if o_j-out < o_j-in then
9          G ← G ∪ o_j ;        // o_j in the initial circle
10         Tab add o_j.ψ;
11     else ∠.addTuple(o_j-in, in, o_j) ;
12 if Tab is full then return G ;
13 sort ∠ by angle in ascending order;
14 foreach tuple (angle, type, o) in ∠ do
15     if type = in then
16         G ← G ∪ o;
17         Add o.ψ to Tab;
18         if Tab is full then return G ;
19     else
20         G ← G \ o;
21         Remove o.ψ from Tab;
22 return ∅;
```



(a) Circle rotation (b) Angles computed

Figure 6: Circle scan procedure w.r.t. query $\{t_1, t_2, t_3, t_4\}$

Example 3: Figure 6 shows an example of the checking process. The outside-in polar angles are marked as blue lines and the inside-out polar angles are marked as red lines. Assume that the rotation reaches angle o_1-out, and before that the keyword-frequency table Tab is $\{t_1{:}2, t_2{:}1, t_4{:}2\}$. Since o_1-out is an object inside-out angle, we remove the keywords covered by o_1 and we obtain Tab = $\{t_1{:}2, t_2{:}1, t_4{:}1\}$. The rotation stops at the next angle o_2-out, and we update Tab as $\{t_1{:}1, t_2{:}1\}$. The next largest angle is o_4-in, and we get Tab = $\{t_1{:}1, t_2{:}2, t_3{:}1\}$. Since Tab still cannot cover all query keywords, we continue the rotation and we reach o_5-in next. Tab is updated as $\{t_1{:}1, t_2{:}2, t_3{:}1, t_4{:}1\}$, and a KEC_q^o is found now and we stop the checking and return the result.

Suppose there are n objects in the sweeping area. The time complexity of sorting the polar angles is $O(n \log n)$. Each outside-in angle and inside-out angle is scanned once during the sweeping. At the initial position the text descriptions of all objects in the sweeping area are read, which has the complexity of $O(n)$, and each subsequent rotation only has complexity of $O(1)$. Hence, the rotation and checking together has complexity $O(n)$, and the total time complexity of this procedure is $O(n \log n)$.

4.3.3 *Approximation ratio and Complexity*

SKECa finds a circle under the error tolerance α. Let G_{skeca} denote the group found by *SKECa*. That is, it is guaranteed that $\phi(\mathsf{MCC}_{G_{skeca}}) \leq \phi(\mathsf{SKEC}_q) + \alpha$. We propose an approach to setting a query-dependent threshold value for α such that we can assure an approximate ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$ for *SKECa*, where ϵ is an arbitrarily small value. The idea is to utilize the result returned by

the 2-approximation algorithm *GKG*. Specifically, after we obtain a group G_{gkg} by invoking *GKG*, we set $\alpha = \epsilon\delta(G_{gkg})/2$, and we have the following lemma.

Theorem 6: The *SKECa* algorithm has an approximation ratio of $\frac{2}{\sqrt{3}} + \epsilon$ by setting $\alpha = \epsilon\delta(G_{gkg})/2$.

PROOF. See Appendix E □

The time complexity is also affected by parameter ϵ. The binary search range sr is $\emptyset(\mathsf{MCC}_{G_{gkg}}) - \delta(G_{gkg})/2 \leq (\frac{2}{\sqrt{3}} - \frac{1}{2})\delta(G_{gkg})$. The binary search stops when the gap between the search upper and lower bounds is smaller than α. Hence, the binary search takes $O(\log(sr/\alpha))$ steps, and it can be computed that $sr/\alpha \leq (\frac{2}{\sqrt{3}} - \frac{1}{2})\delta(G_{gkg})/(\epsilon\delta(G_{gkg})/2) = (\frac{4}{\sqrt{3}} - 1)/\epsilon$.

In the *SKECa* algorithm, in the worst case, the binary search would be performed on all objects in the dataset. Therefore, the time complexity in the worst case is $O(|\mathcal{O}'| \log \frac{1}{\epsilon} n \log n)$, where $|\mathcal{O}'|$ is the number of objects relevant to the query, $O(\log \frac{1}{\epsilon})$ is the steps of binary search performed on an object, and $O(n \log n)$ is the complexity of one step where n is the number of objects in the sweeping area in the worst case. The circleScan() method is invoked only when the sweeping area covers all query keywords on an object. Thus, in practice, the query processing time is much better than analyzed.

4.4 Algorithm *SKECa+*

As analyzed in Section 4.3.2, the cost of checking whether there exists a KEC_q^o with diameter D is determined by the number of objects in the sweeping area which depends on the value of D. In the *SKECa* algorithm, the binary search is performed on each object, and the minimum diameter of the keywords enclosing circle obtained on processed objects is used as the upper bound in subsequent search on remaining objects. Hence, if on the early processed objects the circles found are large, the upper bound is loose for subsequent search and the checking cost is high.

To avoid this problem and improve efficiency, we propose to perform binary search on all objects in $\mathcal{O}'$ together, instead of on each of them separately. Specifically, we find the diameter and the position of SKEC_q directly using binary search, instead of first finding all object-across smallest keywords enclosing circles and then selecting the best one. We call the process enhanced binary search.

We set the range of the enhanced binary search as follows. Given a query q, if there exists an o-across keywords enclosing circle with diameter D on any object o in $\mathcal{O}'$, D is an upper bound of $\emptyset(\mathsf{SKEC}_q^o)$, and obviously it is also an upper bound of $\emptyset(\mathsf{SKEC}_q)$. If on an object o there exists no o-across keywords enclosing circle with diameter D, D is a lower bound of $\emptyset(\mathsf{SKEC}_q^o)$; if D is a lower bound of $\emptyset(\mathsf{SKEC}_q^o)$ for every o in $\mathcal{O}'$, D is a lower bound of $\emptyset(\mathsf{SKEC}_q)$. The enhanced algorithm is presented in Algorithm 3.

We obtain $\mathcal{O}'$ and set the upper and lower bounds as we do in Algorithm 1. It can be proven that $\delta(G_{gkg})/2$ is also a lower bound of SKEC_q by following the proof in Lemma 1. On each object o, we use array $maxInvalidRange$ to record the largest diameter such that there exists no o-across keywords enclosing circle with diameter $maxInvalidRange[o]$ according to the previous checks. Initially, $maxInvalidRange[o]$ is set to 0 (line 7). The value is used to avoid unnecessary checking on an object o.

The enhanced binary search is applied on all objects together (lines 9-24). On each object o in $\mathcal{O}'$, before we find SKEC_q^o with diameter $diam$, we first compare whether $diam$ is less than the invalid diameter stored in $maxInvalidRange$ to avoid unnecessary checking. We can safely discard o if $diam$ is smaller than $maxInvalidRange[o]$ according to Property 1 (lines 12–13). If we can find a keywords enclosing circle KEC_q^o with diameter $diam$, we update the upper bound of SKEC_q and the current best circle,

Algorithm 2: *SKECa+* (q, α)

```
1  G_gkg ← GKG (q); C_cur ← MCC_{G_gkg};
2  O' ← objects containing at least one keyword in q;
3  searchUB ← ø(C_cur);
4  searchLB ← δ(G_gkg)/2;
5  foreach object o in O' do
6      if o.ψ = q then return {o} ;
7      maxInvalidRange[o] ← 0;
8  while searchUB − searchLB > α do
9      diam ← (searchUB + searchLB)/2;
10     foundResult ← False;
11     foreach o in O' do
12         if diam < maxInvalidRange[o] then
13             continue;
14         oskec ← circleScan(o, diam);
15         if oskec ≠ ∅ then
16             searchUB ← diam;
17             C_cur ← oskec;
18             foundResult ← True;
19             break;
20         else
21             if diam > maxInvalidRange[o] then
22                 maxInvalidRange[o] ← diam;
23     if foundResult = False then
24         searchLB ← diam;
25 G_skeca ← objects in C_cur;
26 return G_skeca;
```

and we terminate the checking using $diam$ immediately (lines 15–19). Otherwise, we update the maximum invalid diameter of o if it is smaller than $diam$, because no o-across keywords enclosing circle with diameter $diam$ exists (lines 20–22). If on all objects we fail in finding the keywords enclosing circle with diameter $diam$, we increase the lower bound for subsequent search (lines 23–24).

Approximation ratio and Complexity. Since *SKECa+* finds the same circle as found in *SKECa*, the result group is the same as that returned by *SKECa*, and thus this algorithm also has an approximation ratio of $(\frac{2}{\sqrt{3}} + \epsilon)$.

SKECa+ performs binary search on all objects relevant to the query together. The bounds for binary search are the same as those used in *SKECa*, and thus the steps of binary search is also $O(\log \frac{1}{\epsilon})$. *SKECa+* also invokes the procedure circleScan(), which has complexity $O(n \log n)$. In the worst case, the binary search is performed on all objects as well. Therefore, the worst case time complexity of *SKECa+* is also $O(|\mathcal{O}'| \log \frac{1}{\epsilon} n \log n)$. However, after we know a keywords enclosing circle with diameter $diam$ exists on an object in *SKECa+*, we can immediately stop the search using $diam$ and avoid the checking on the remaining objects. Therefore, it has much better efficiency in practice than does *SKECa*, which is to be shown in the experimental study in Section 6.

The number of objects n in the sweeping area w.r.t. an object depends on the query. If the query keywords are frequent and the optimal result has a large diameter, n would be large. We study the effect of both the frequencies of query keywords and the diameter of the optimal result on the efficiency of our algorithms in the experiments.

5. EXACT ALGORITHM

It is challenging to develop an efficient exact algorithm for mCK queries, as an exact algorithm cannot avoid an exhaustive search in the object space. The best known solution [22] performs exhaustive search on all objects that contain at least one query keyword, and thus the computational cost is high if such objects are large in number. In this section, we propose an exact algorithm, which

leverages the result of our approximation algorithm *SKECa+* to greatly reduce the space of exhaustive search.

5.1 Algorithm Framework

If we know the smallest circle that encloses the optimal group G_{opt}, i.e., $\mathsf{MCC}_{G_{opt}}$, we can do exhaustive search only on objects in $\mathsf{MCC}_{G_{opt}}$ and the search space can be reduced significantly, compared to the search space of the best known algorithm [22]. However, we know neither the radius nor the center of $\mathsf{MCC}_{G_{opt}}$. Note that the circle $\mathsf{MCC}_{G_{opt}}$ is different from the smallest keywords enclosing circle w.r.t. q, i.e., SKEC_q. For example, as shown in Figure 2, given a query $q = \{t_1, t_2, t_3\}$, SKEC_q is MCC_{G_1} , but the optimal group G_2 is enclosed in MCC_{G_2}.

Although we do not know the exact size of $\mathsf{MCC}_{G_{opt}}$, we prove that the diameter of $\mathsf{MCC}_{G_{opt}}$ can be bound by the diameter of SKEC_q as follows.

Lemma 2: $\phi(\mathsf{MCC}_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\phi(\mathsf{SKEC}_q)$.

PROOF. Denote the group enclosed in SKEC_q by G_{skec}. According to Theorem 4, $\phi(\mathsf{MCC}_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$, and $\phi(\mathsf{SKEC}_q) \geq \delta(G_{skec})$. Because G_{opt} has the smallest diameter, we have $\phi(\mathsf{MCC}_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt}) \leq \frac{2}{\sqrt{3}}\delta(G_{skec}) \leq \frac{2}{\sqrt{3}}\phi(\mathsf{SKEC}_q)$. □

With Lemma 2, we propose to utilize SKEC_q to reduce the exhaustive search space. Our basic idea is that, we find the circles that might be $\mathsf{MCC}_{G_{opt}}$ utilizing SKEC_q. Within each such candidate circle, we perform an exhaustive search (with some pruning strategies) on the objects enclosed by it to find a group. After all candidate circles are found and checked, the best group found must be the optimal group. Note that we use *SKECa+* to find approximate SKEC_q, rather than *SKEC* to find the exact SKEC_q, which is computationally expensive. The circle returned by *SKECa+* has a diameter not exceeding $\phi(\mathsf{SKEC}_q) + \alpha$. Thus, $\phi(\mathsf{MCC}_{G_{opt}})$ is bound by $\frac{2}{\sqrt{3}}$ times the diameter of the circle returned by *SKECa+* as well.

We proceed to explain how to find the candidate circles that might be $\mathsf{MCC}_{G_{opt}}$. Since $\mathsf{MCC}_{G_{opt}}$ is the smallest circle that encloses the optimal group G_{opt}, there must exist an object in G_{opt} on the boundary of $\mathsf{MCC}_{G_{opt}}$. The problem is that we do not know this object, which can be any object o containing at least one query keyword. To avoid finding candidate circles on each such object o, we establish the following lemma to filter out objects that cannot be on the boundary of $\mathsf{MCC}_{G_{opt}}$. We denote by G_{skeca} the group found by *SKECa+* and the smallest circle enclosing G_{skeca} by $\mathsf{MCC}_{G_{skeca}}$.

Lemma 3: On an object o, if the diameter of SKEC_q^o is larger than $\frac{2}{\sqrt{3}}$ times the diameter of $\mathsf{MCC}_{G_{skeca}}$, o cannot be on the boundary of $\mathsf{MCC}_{G_{opt}}$.

PROOF. If o is on the boundary of $\mathsf{MCC}_{G_{opt}}$, $\phi(\mathsf{SKEC}_q^o) \leq \phi(\mathsf{MCC}_{G_{opt}})$ (note that even though an object o is indeed the object on the boundary of $\mathsf{MCC}_{G_{opt}}$, SKEC_q^o may still not be $\mathsf{MCC}_{G_{opt}}$). Because $\phi(\mathsf{MCC}_{G_{opt}}) \leq \frac{2}{\sqrt{3}}\phi(\mathsf{SKEC}_q)$ (according to Lemma 2), and $\phi(\mathsf{MCC}_{G_{skeca}}) \geq \phi(\mathsf{SKEC}_q)$ (because of the binary search), we have $\phi(\mathsf{SKEC}_q^o) \leq \frac{2}{\sqrt{3}}\phi(\mathsf{MCC}_{G_{skeca}})$, leading to a contradiction. □

With Lemma 3, we prune some objects from consideration for finding candidate circles as follows: Recall that in *SKECa+*, we use an array $maxInvalidRange$ to store the maximum diameter on each object o such that there exists no KEC_q^o with diameter $maxInvalidRange[o]$. This means that $\phi(\mathsf{SKEC}_q^o)$ must be larger than $maxInvalidRange[o]$. Hence, in the exact algorithm, we can discard o if $maxInvalidRange[o] \geq \frac{2}{\sqrt{3}}\phi(\mathsf{MCC}_{G_{skeca}})$ according to Lemma 3. If an object o cannot be pruned, o is possibly on the boundary of $\mathsf{MCC}_{G_{opt}}$, and we find all candidate circles that cover all query keywords and pass through o, and in each of them we perform an exhaustive search.

Algorithm 3: *EXACT* (q, α)

```
1 G_skeca ← SKECa+ (q, α);
2 diam ← (2/√3)ø(MCC_{G_skeca});
3 bestGroup ← G_skeca;
4 get the array maxInvalidRange computed in SKECa+;
5 foreach o in O' do
6     if maxInvalidRange[o] < diam then
7         circleScanSearch(o, diam, bestGroup);
8 return bestGroup;
```

The pseudocode is described in Algorithm 3. We first invoke *SKECa+* to get an approximate result (line 1). Then we compute the upper bound $diam$ of the diameter of the smallest circle that encloses the optimal group (line 2). On each object o, if it cannot be pruned (line 6), we invoke Procedure circleScanSearch() to find all keywords enclosing circles with diameter $diam$ around o, to do exhaustive search in each such circle, and to update the best group (line 7). After all objects in $\mathcal{O}'$ are processed, we return $bestGroup$ as the result.

Procedure circleScanSearch(). In this procedure, we first find all o-across keywords enclosing circles with diameter $\frac{2}{\sqrt{3}}\phi(\mathsf{MCC}_{G_{skeca}})$ on an object o, which are candidate circles. The idea is similar to that of Procedure circleScan(). We first fix a sweeping area around o that contains all objects relevant to the query whose distances to o are smaller than $\frac{2}{\sqrt{3}}\ \phi(\mathsf{MCC}_{G_{skeca}})$. In this sweeping area, we rotate the circle around o with diameter $\frac{2}{\sqrt{3}}\ \phi(\mathsf{MCC}_{G_{skeca}})$ clockwise. We also use a table Tab to store all keywords with their frequencies covered in the rotating circle, and update it once an object is rotated inside-out or outside-in. Once all query keywords are contained in Tab, we know that a KEC_q^o (which is a candidate circle) is found.

In circleScan(), once a KEC_q^o is found it returns immediately, because the procedure only checks if there exists a KEC_q^o with a given diameter. In circleScanSearch() we need to perform an exhaustive search to find the best group in the circle. Hence, after the group is found, we still need to repeat the above process to find all candidate circles on o, and to enumerate the best group in each of them. We invoke Procedure search() to perform exhaustive search to enumerate the group with the smallest diameter in a candidate circle, which is presented as follows.

5.2 Procedure search()

The search() procedure is designed as a branch-and-bound process. We adopt the depth-first-search strategy to do enumeration. We use $selectedSet$ to store the objects that are already selected in the current enumeration. The object o on which the search is performed is always in $selectedSet$ since it must be contained in the generated group. We use $candidateSet$ to store the objects in the circle that are possible to combine with objects in $selectedSet$ to form a group whose diameter is smaller than that of the current best group.

In each step, we select one object from $candidateSet$ and check if combining it with the objects in $selectedSet$ can generate a better group. If so, we remove this object from $candidateSet$ and add it to $selectedSet$. This step is performed iteratively. The level of this depth-first-search is at most the number of query keywords, because each enumerated object contains at least one new query keyword. The current best group $curGroup$ is updated when a group covering all the query keywords with smaller diameter is found.

The search complexity is exponential with the number of objects relevant to the query in a candidate circle. We develop several pruning strategies utilizing both textual and spatial properties in search() to improve efficiency.

Pruning Strategy 1. Given an object o, and let G denote the group $selectedSet \cup o$, if the diameter of G exceeds the diameter of the current best group $curGroup$, o does not need to be added to $selectedSet$. This is because that, for any group G' generated from G, it is true that $\delta(G') \geq \delta(G)$. Hence, the diameter of G' is also larger than $\delta(curGroup)$, which means that no better groups can be obtained from G.

Pruning Strategy 2. If an object o cannot contribute any new keyword to $selectedSet$, o is not necessary to be added to $selectedSet$. This can be justified as follows: Denote the group $selectedSet \cup o$ by G. $selectedSet$ and G cover the same set of query keywords and $\delta(selectedSet) \leq \delta(G)$, and thus G can be pruned.

Pruning Strategy 3. If the objects in $candidateSet$ cannot cover the keywords that have not be covered by $selectedSet$, we can stop the search using $selectedSet$. This is because that even if we select all the objects in $candidateSet$, and combine them with the objects in $selectedSet$, a group covering all the query keywords cannot be generated.

Procedure search(q, $selectedSet$, $candidateSet$, $maxId$)

```
1  if selectedSet.ψ = q then
2      if δ(selectedSet) ≤ δ(curGroup) then
3          curGroup ← selectedSet;
4      return curGroup;
5  if δ(selectedSet) > δ(curGroup) then return ∅ ;
6  nextSet ← ∅;
7  leftKeyowrds ← ∅;
8  foreach candidate object o_c in candidateSet do
9      if δ(selectedSet ∪ o_c) > δ(curGroup) then continue ;
10     if (q − selectedSet.ψ) ∩ o_c.ψ = ∅ then continue ;
11     if o_c.Id < maxId then continue ;
12     nextSet ← nextSet ∪ o_c;
13     leftKeywords ← leftKeywords ∪ o_c.ψ;
14 if leftKeywords ∪ selectedSet.ψ! = q then
15     return curGroup;
16 foreach object o_n in nextSet do
17     newselSet ← selectedSet ∪ o_n;
18     newcandSet ← nextSet \ o_n;
19     group ← search(q, newselSet, newcandSet, o_n.Id);
20     if δ(group) < δ(curGroup) then
21         curGroup ← group;
22 return curGroup;
```

In Procedure search(), if $selectedSet$ already covers all query keywords, we compare this group with the current best group, and we return the better one as the result (lines 1–4). We then check if the selected group is already worse than the current best solution (line 5). In lines 8–13, we scan $candidateSet$ and filter out objects that cannot be combined with objects in $selectedSet$. Line 9 applies the pruning strategy 1, line 10 applies the pruning strategy 2, and line 11 is used to avoid enumerating duplicate groups. We check whether the pruning strategy 3 is satisfied in lines 14–15. After we add a new object to $selectedSet$, we invoke the procedure recursively to find the group with newly selected object set $newselSet$ and new candidate object set $newcandSet$ and update $curGroup$ correspondingly (lines 17–21).

Complexity. If there are relevant n objects in the sweeping area in circleScanSearch() in the worst case, the complexity of computing and sorting the rotation angles is $O(n \log n)$. The exhaustive search in the sweeping area has complexity $O(n^{|q|-1})$, because the depth-first search level is at most $|q|$ and the object on which the search is performed already contains at least one query keyword. Since the diameter of the candidate circle is bounded as shown in Lemma 2, n is usually not large. In summary, on an object the total search complexity is $O(n \log n + n^{|q|-1}) \approx O(n^{|q|-1})$. If there are $\mathcal{O}'$ objects relevant to the query, the worst case time complexity of *EXACT* is $O(|\mathcal{O}'|n^{|q|-1})$, where in general $n \ll |\mathcal{O}'|$. In practice, we only do the search on objects satisfying the bound as described in Lemma 3, and thus the practical performance is better than analyzed. Note that the worst case time complexity of the best known solution [22] is $O(|\mathcal{O}'|^{|q|})$, which it worse than *EXACT*. As to be shown in the experimental study in Section 6, *EXACT* is much more efficient.

6. EXPERIMENTAL STUDY

6.1 Experimental Settings

Algorithms. We evaluate four approximation algorithms, namely *GKG* (in Section 3), *SKEC* (in Section 4.2), *SKECa* (in Section 4.3), and *SKECa+* (in Section 4.4), and the exact algorithm *EXACT* (in Section 5). We also compare our exact algorithm with the state-of-the-art solution proposed in the work [22], denoted by *VirbR*, and the methods of adapting the spatial group keyword query [16], denoted by *ASGK* and *ASGKa* (Section 2.2).

Dataset	Number of Objects	Unique words	Total words
NY	485,059	116,546	1,143,013
LA	724,952	161,489	1,833,486
TW	1,000,100	487,552	5,170,495

Table 1: Dataset properties

Datasets. We use three real-life datasets. Table 1 lists some properties of these datasets. Datasets NY and LA are crawled using Google Place API in New York and Los Angeles, respectively. Each crawled object has a name and a type such as "food" and "restaurant," used as the textual description of the object, and a pair of latitude and longitude representing its location. Dataset TW is crawled from Twitter within the area of USA. Each geo-tweet is treated as a geo-textual object, and its content is used as the description of the object and its latitude and longitude are used as the geo-location of the object.

The location of an object is in form of a pair of latitude and longitude. In order to compute the Euclidean distance between locations, we convert the data to the UTM (Universal Transverse Mercator coordinate system) format, using World Geodetic System 84 specification.

Query generation. We generate 5 query sets with different numbers of keywords, i.e., 2, 4, 6, 8, and 10, for each dataset. Each set comprises 50 queries. According to the complexity analysis of the proposed approximation and exact algorithms, their runtimes are affected by the diameter of the optimal group w.r.t. a given query. When evaluating the effect of a certain parameter, we try to bound the diameter of the optimal group for a query, so that the effect of the diameter does not vary too much for queries in one set. For example, to set the upper bound diameter at 20% of the diameter of the whole dataset, we first randomly draw a circle with diameter no larger than 20% of the diameter of all objects in the dataset, and then we randomly select the terms that appear in this circle according to their frequencies. This makes sure that the diameter of the optimal group cannot exceed 20% of the diameter of the whole dataset. It is hard to impose a lower bound constraint when generating queries. We also study the effect of the diameter bound on the efficiency of algorithms.

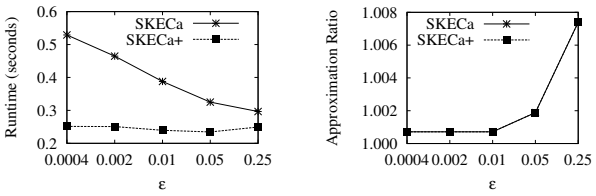
We set the number of query keywords to 6 and the diameter bound to 20% of the diameter of a dataset by default in our experiments.

Setup. The virtual bR*-tree index structure is disk resident, and the page size is set to 4KB. The number of children of a node in the tree is set to 100. All algorithms were implemented in C++ and run on Linux with a 2.66GHz CPU and 8GB RAM.

6.2 Experimental Results

6.2.1 Tuning the binary search parameter ϵ

The value of ϵ affects both efficiency and accuracy of *SKECa* and *SKECa+*. We vary ϵ from 0.0004 to 0.25. Figures 9(a) and 9(b) show the runtime and the accuracy of *SKECa* and *SKECa+* when we vary ϵ on the LA dataset, respectively.



(a) Runtime (LA) (b) Appro. Ratio (LA)

Figure 7: Varying ϵ

It can be observed that *SKECa* always runs slower than *SKECa+*. Recall that the steps of binary search needed to be performed is linear with $O(\log \frac{1}{\epsilon})$ in *SKECa*. Hence, as ϵ increases, *SKECa* runs faster because a larger ϵ reduces the steps of binary search performed on each object. *SKECa* and *SKECa+* return the same result, and the difference is the way of performing the binary search. Parameter ϵ also determines the steps of binary search in *SKECa+*. However, since *SKECa+* performs the binary search on all objects relevant to the query together, its runtime is only slightly affected and is much better than that of *SKECa*.

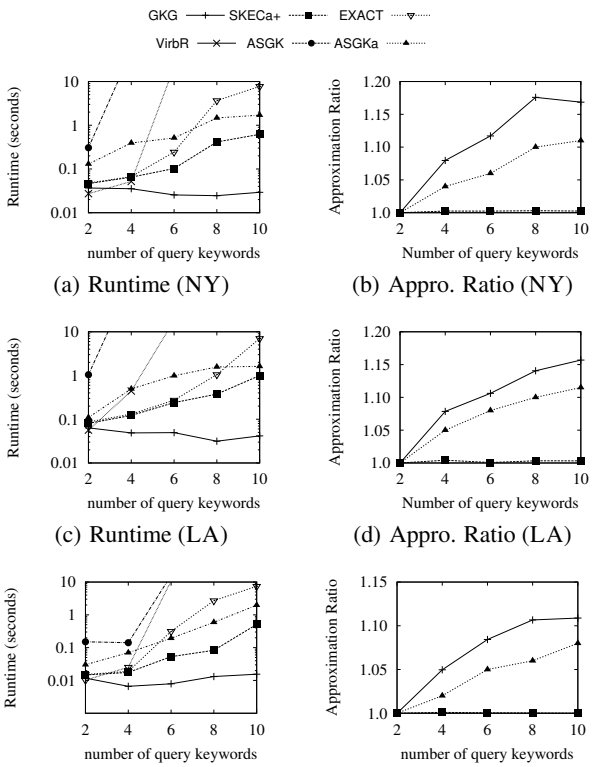
The two algorithms have the same accuracy, and the accuracy drops as ϵ becomes larger. Using a smaller ϵ can obtain a circle that is more close to SKEC_q, and thus the group enclosed in the circle found finally is more close to the optimal result. This is consistent with the ratio of the two algorithms, i.e., $(\frac{2}{\sqrt{3}} + \epsilon)$.

Because *SKECa+* outperforms *SKECa* consistently, we report the results of *SKECa+* in subsequent experiments only. Based on Figure 6, we set ϵ to 0.01 for *SKECa+* by default, because it strikes a good tradeoff between accuracy and efficiency. Similar results are observed on the other two datasets, and we report them in Appendix F.

6.2.2 Varying the number of query keywords

Figure 8 shows the runtime (in logarithmic scale) and accuracy of the six algorithms, i.e., *GKG*, *SKECa+*, *EXACT*, *VirbR*, *ASGK* and *ASGKa* when we vary the number of query keywords on each dataset.

On all datasets, *SKECa+* achieves better accuracy than does *GKG*, because it has a better approximation ratio. *SKECa+* can always obtain nearly optimal groups. *GKG* runs the fastest on all datasets. It is interesting to observe that *GKG* runs faster as the number of query keywords increases on some datasets. The reason is as follows. *GKG* only searches for a group of objects that contain the most infrequent query keyword. Given more query keywords, the most infrequent keyword is more likely to have lower frequency, and thus fewer objects need to be checked. *SKECa+* runs slower as the number of query keywords increases. Recall that the complexity of *SKECa+* is $O(|\mathcal{O}'| \log \frac{1}{\epsilon} n \log n)$. Given more query keywords, the number of objects relevant to the query $|\mathcal{O}'|$ becomes larger. In addition, the number of relevant objects in the sweeping area n in Procedure circleScan() also increases.



(a) Runtime (NY) (b) Appro. Ratio (NY)

(c) Runtime (LA) (d) Appro. Ratio (LA)

(e) Runtime (TW) (f) Appro. Ratio (TW)

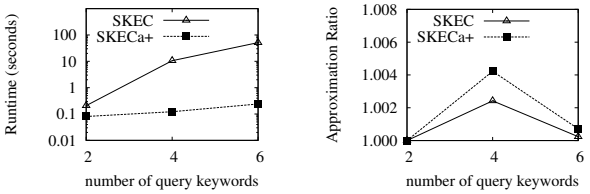
Figure 8: Varying number of query keywords

EXACT outperforms *VirbR* by more than an order of magnitude on all datasets when queries contain more than 4 keywords. *EXACT* can answer most queries within 10 seconds, but *VirbR* takes several minutes to answer a query in average when queries contain 8 or 10 keywords. Recall that the complexity of *EXACT* is $O(|\mathcal{O}'|n^{|q|-1})$. As the number of query keywords increases, the number of objects relevant to the query $|\mathcal{O}'|$ increases, and the number of objects in the sweeping area n also increases, and hence *EXACT* runs slower. The complexity of *VirbR* is $O(|\mathcal{O}'|^{|q|})$, and thus it also runs slower as the number of query keywords increases. Recall that the *EXACT* algorithm first invokes *SKECa+* to reduce the search space then it performs the exhaustive search. The experimental results show that *SKECa+* is able to prune the search space significantly, thus making *EXACT* efficient.

We observe that the performance of *ASGK* is much worse than *VirbR* and *EXACT*. Both the efficiency and accuracy of *ASGKa* are much worse than *SKECa+*. They are not originally developed to process the mCK query, and the results show the adaption of the algorithms [16] for processing the mCK query is not efficient. We will ignore *ASGK* and *ASGKa* in the subsequent experiments.

Figure 9 shows the comparison of the algorithms *SKEC* and *SKECa+* on dataset LA. The runtime of *SKECa+* is much better than *SKEC*, which is consistent with the analysis of the time complexity of the two algorithms in Section 4. *SKEC* is extremely slow when the number of query keywords is large, because the number of relevant objects ($\mathcal{O}'$) becomes larger, and the worst case complexity of this algorithm is O($|\mathcal{O}'|^4$). When there are more than 6 keywords, lots of queries take at least 5 minutes to finish, and thus we only report the results on query sets containing 2, 4, and 6 keywords to make the figure readable. It can be observed that the two algorithms have similar accuracy, because we set ϵ to 0.01, a very

small value to control the error of *SKECa+*. The relative performance comparisons between *SKEC* and *SKECa+* are qualitatively similar in other experiments and we do not report them.



(a) Runtime (LA) (b) Appro. Ratio (LA)

Figure 9: Comparing *SKEC* with *SKECa+*

6.2.3 *Varying the optimal group diameter bound*

In this set of experiments, we study the effect of the diameter bound of the optimal group for a query. We vary the diameter bound from 10% to 30% of the diameter of a dataset. Figure 10 shows the results on LA and TW.

Figures 10(a) and 10(e) show the runtime of two approximation algorithms on LA and TW. *GKG* searches for a group around each object containing the most infrequent keyword, and the group diameter does not affect the runtime of *GKG*. As the diameter bound increases *SKECa+* runs slower. The reason is that with a larger bound the optimal group for a query is more likely to have a larger diameter, and *SKECa+* needs to scan a larger sweeping area to find a keywords enclosing circle with the given diameter. From Figures 10(b) and 10(f) we can see that *SKECa+* has better accuracy than *GKG* and is always able to achieve nearly optimal results.

Due to the hardness of answering mCK queries, the exact algorithms may be very slow on some queries which dominate the average running time. For the readability of our figures, we set a timeout threshold to 1 minute. We observe that the queries where *VirbR* succeeds to find the result within the timeout threshold can always be answered by our algorithm *EXACT*. When comparing the two algorithms, we only report on queries where both algorithms succeed to return a result within the time limit.

Figures 10(c) and 10(g) show the runtime of *EXACT* and *VirbR* (y-axis in logarithmic scale) on LA and TW. It is shown that *EXACT* outperforms *VirbR* by about one order of magnitude on those queries that both algorithms can finish in 1 minute. The success rate of two algorithms is shown in Figures 10(d) and 10(h). *EXACT* always has a better success rate (close to 100%). Both algorithms have lower success rate as the diameter bound increases. This is because that if the optimal group has a larger diameter the exhaustive search takes longer time in both algorithms. We observe similar result on NY and ignore it.

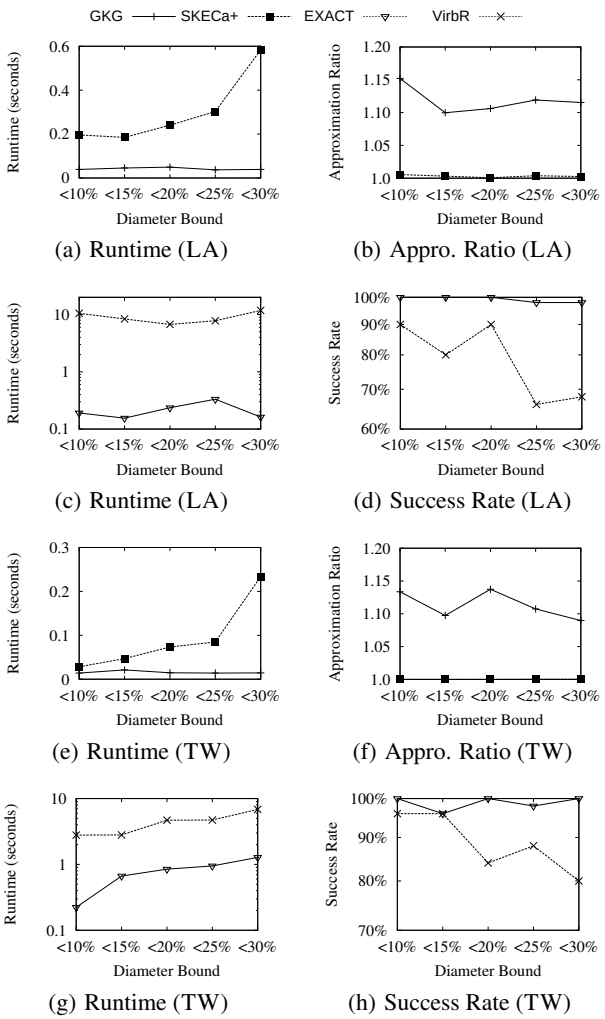
To further study these slow queries, Figure 11 compares the runtime and success rate of *EXACT* and *VirbR*, when the timeout threshold varies from 15 seconds to 4 minutes and the diameter is bounded by 30% of the whose space. *EXACT* solves most queries within 15 seconds and it always outperforms *VirbR*.

We study the cases where *EXACT* greatly outperforms *VirbR*, and we find that the result groups of these queries have small diameters. Recall that *EXACT* first reduces the search space by utilizing *SKECa+* and then performs the exhaustive search in the reduced space. For queries that cannot be solved within 15 seconds by both algorithms, we find that these queries contain keywords with both high and low frequency and the diameter of the result group is large.

6.2.4 *Varying the query keywords frequencies*

In this set of experiments, we vary the frequencies of query keywords and evaluate the performance of five algorithms on LA, i.e., *GKG*, *SKECa+*, *EXACT*, and *VirbR*. We rank terms in ascending order of their frequencies, and then generate a query set using lower $x\%$ terms, i.e., we select terms from the $x\%$ least frequent terms to form a query. We vary $x\%$ from 20% to 100% to generate 5 query sets (100% means that the query keywords are selected from all terms in a dataset according to their frequency, as we do in previous experiments).

Figure 12 shows the runtime and accuracy of four algorithms. It can be observed in Figure 12(a) that as the frequency of query keywords increases, both approximation algorithms run slower. This is because that more objects need to be taken into consideration during algorithm execution. The runtime of *EXACT* and *VirbR* is reported only on queries that can be answered within the 1 minute threshold. *EXACT* has better success rate as shown in Figure 12(d); on the queries that both algorithms succeed, EXACT is almost one



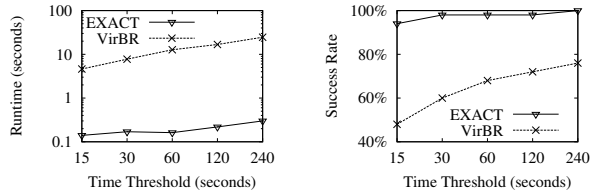
(a) Runtime (LA) (b) Appro. Ratio (LA)

(c) Runtime (LA) (d) Success Rate (LA)

(e) Runtime (TW) (f) Appro. Ratio (TW)

(g) Runtime (TW) (h) Success Rate (TW)

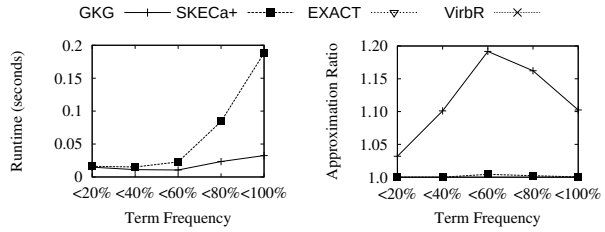
Figure 10: Varying optimal group diameter bound



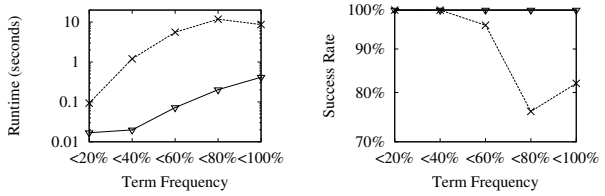
(a) Runtime (LA) (b) Success Rate (LA)

Figure 11: Varying timeout threshold

order of magnitude faster than *VirbR*, as shown in Figure 12(c) (y-axis in logarithmic scale). *EXACT* and *VirbR* run slower as query keywords become more frequent, which is consistent as analyzed. We observe similar results on the other two datasets and they are not reported.



(a) Runtime (LA) (b) Appro. Ratio (LA)

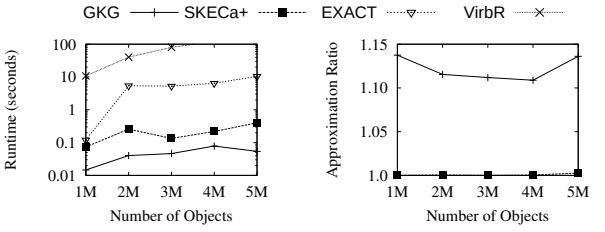


(c) Runtime (LA) (d) Success Rate (LA)

Figure 12: Varying query keywords frequencies

6.2.5 *Scalability*

To evaluate scalability, we use 5 datasets containing tweets with locations, all of which are crawled from Twitter. The largest dataset contains 5 million tweets, and we sample other datasets from it. Figure 13 shows the runtime and approximation ratio of four algorithms on TW, i.e., *GKG*, *SKECa+*, *EXACT* and *VirbR* (the number of query keywords is 6). Both approximation algorithms scale quite well with the size of the dataset, and all queries can be answered within 1 second by *GKG* and *SKECa+*. The *EXACT* algorithm also scales well. *VirbR* runs slower than *EXACT* by orders of magnitude, and it takes more than one minute to answer a query in average when the dataset contains more than 3 million objects. The accuracy changes only slightly, and *SKECa+* always returns nearly optimal results and has better accuracy than *GKG*.



(a) Runtime (b) Appro. Ratio

Figure 13: Scalability

7. RELATED WORK

The spatial keyword queries are gaining in prevalence and are widely used in many real-life applications. For example, in Google Maps "search nearby" offers users the functionality to retrieve points of interest around a specified location. Most existing studies on spatial keyword querying focus on retrieving a list of single geo-textual objects such that each object returned is both relevant to query keywords and close to query location. The spatial keyword queries can be categorized into two types according to how keywords are used in querying. In some proposals (e.g., [4, 6, 9, 13, 20, 23]), keywords are used as Boolean predicates to filter out objects that do not contain the keywords, and the remaining objects are ranked based on their spatial proximity to the query. In other proposals(e.g., [6, 7, 14]), spatial proximity and textual relevance are combined by a linear function to rank geo-textual objects.

Several recent proposals consider searching for a group of geo-textual objects instead of single objects. As described in Section 2.2, the mCK query is proposed and studied in the works [21, 22]. The existing studies on the mCK query [21, 22] do not study the hardness of this problem and propose exact algorithms only. We prove that answering the mCK query is NP-hard and propose both exact and approximation algorithms. The spatial group keyword query SGK [2, 16] takes a location and a set of keywords as query arguments. It retrieves a group that covers all the query keywords, has a small diameter, and is close to the query location. As analyzed in Section 2.2, one special case of the SGK query [2] is equivalent to the mCK query, and the algorithm proposed is reduced to the method [21] when being applied to the mCK query, and thus is worse than the baseline *VirbR* [22], used in our experiments. As discussed in Section 2.2, another type of the SGK query [16] can be adapted to answer the mCK query, and as shown in the experimental study, its performance is much worse than our proposed algorithm. The SGK query and the mCK query suit different application scenarios. Consider the tourist example in Section 1. If the tourist has already booked a hotel, then the SGK query could be used where the hotel serves as the query location. Otherwise, the mCK query is suitable where "hotel" is one of the query keywords.

In addition, some proposals consider spatial keyword queries on road networks (e.g., [3, 18]), and spatial keyword queries on trajectory databases are also studied (e.g., [8, 19, 24]). There also exists some work (e.g., [12]) querying geo-textual objects whose locations are represented by rectangles. The problem of spatio-textual similarity joins is also studied to join two sets of geo-textual objects [1, 15].

8. CONCLUSION

We study the problem of answering mCK queries in this paper. We prove that this problem is NP-hard. We propose a 2-approximation greedy approach as a baseline. Utilizing this greedy method, we first devise an approximation algorithm *SKEC* that aims at finding the smallest circle that can enclose a group of objects covering all query keywords. We prove that its approximation ratio is $\frac{2}{\sqrt{3}}$. *SKEC* has a high complexity, and we design another two approximation algorithms, *SKECa* and *SKECa+*, to find such a circle approximately for better efficiency. Their approximation ratio is $(\frac{2}{\sqrt{3}} + \epsilon)$, where ϵ can be an arbitrarily small positive value. We also design an exact algorithm utilizing *SKECa+* to reduce the exhaustive search space significantly. Extensive experiments were conducted, which verifies our theoretical analysis and shows that our exact algorithm outperforms the best known solution by an order of magnitude. In the future, it would be of interest to investigate the problem of answering the mCK query in a distributed setting.

9. ACKNOWLEDGEMENTS

This research was carried out at the Rapid-Rich Object Search (ROSE) Lab at the Nanyang Technological University, Singapore. The ROSE Lab is supported by the National Research Foundation, Prime Minister's Office, Singapore, under its IDM Futures Funding Initiative and administered by the Interactive and Digital Media Programme Office. This work was also supported in part by a Singapore MOE AcRF Tier 2 Grant (ARC30/12). We thank the authors [16, 22] for sharing their codes.

10. REFERENCES

[1] P. Bouros, S. Ge, and N. Mamoulis. Spatio-textual similarity joins. *PVLDB*, 6(1):1–12, 2012.

[2] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384, 2011.

[3] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of interest for user exploration. *Proceedings of the VLDB Endowment*, 7(9), 2014.

[4] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *SSDBM*, pages 87–95, 2010.

[5] L. Chen, G. Cong, C. S. Jensen, and D. Wu. Spatial keyword query processing: An experimental evaluation. *PVLDB*, 6(3):217–228, 2013.

[6] M. Christoforaki, J. He, C. Dimopoulos, A. Markowetz, and T. Suel. Text vs. space: efficient geo-search query processing. In *CIKM*, pages 423–432, 2011.

[7] G. Cong, C. S. Jensen, and D. Wu. Efficient retrieval of the top-k most relevant spatial web objects. *PVLDB*, 2(1):337–348, 2009.

[8] G. Cong, H. Lu, B. C. Ooi, D. Zhang, and M. Zhang. Efficient spatial keyword search in trajectory databases. *arXiv preprint arXiv:1205.2880*, 2012.

[9] I. De Felipe, V. Hristidis, and N. Rishe. Keyword search on spatial databases. In *ICDE*, pages 656–665, 2008.

[10] D. Elzinga and D. Hearn. The minimum covering sphere problem. *Management Science*, 19(1):96–104, 1972.

[11] J. Elzinga and D. W. Hearn. Geometrical solutions for some minimax location problems. *Transportation Science*, 6(4):379–394, 1972.

[12] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu. SEAL: Spatio-textual similarity search. *PVLDB*, 5(9):824–835, 2012.

[13] R. Hariharan, B. Hore, C. Li, and S. Mehrotra. Processing spatial-keyword (SK) queries in geographic information retrieval (GIR) systems. In *SSDBM*, page 16, 2007.

[14] Z. Li, K. C. K. Lee, B. Zheng, W.-C. Lee, D. L. Lee, and X. Wang. IR-Tree: An efficient index for geographic document search. *TKDE*, 23(4):585–599, 2011.

[15] S. Liu, G. Li, and J. Feng. Star-join: spatio-textual similarity join. In *CIKM*, pages 2194–2198, 2012.

[16] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries: a distance owner-driven approach. In *SIGMOD*, pages 689–700, 2013.

[17] N. Megiddo. Linear-time algorithms for linear programming in R^3 and related problems. In *FOCS*, pages 329–338, 1982.

[18] J. B. Rocha-Junior and K. Nørvåg. Top-k spatial keyword queries on road networks. In *EDBT*, pages 168–179, 2012.

[19] S. Shang, R. Ding, B. Yuan, K. Xie, K. Zheng, and P. Kalnis. User oriented trajectory search for trip recommendation. In *EDBT*, pages 156–167, 2012.

[20] C. Zhang, Y. Zhang, W. Zhang, and X. Lin. Inverted linear quadtree: Efficient top-k spatial keyword search. In *ICDE*, pages 901–912, 2013.

[21] D. Zhang, Y. M. Chee, A. Mondal, A. K. H. Tung, and M. Kitsuregawa. Keyword search in spatial databases: Towards searching by document. In *ICDE*, pages 688–699, 2009.

[22] D. Zhang, B. C. Ooi, and A. K. H. Tung. Locating mapped resources in Web 2.0. In *ICDE*, pages 521–532, 2010.

[23] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. In *EDBT*, pages 359–370, 2013.

[24] K. Zheng, S. Shang, N. J. Yuan, and Y. Yang. Towards efficient search for activity trajectories. In *ICDE*, pages 230–241, 2013.

APPENDIX

A. PROOF OF THEOREM 1

PROOF. We prove the theorem by a reduction from the 3-SAT problem. An instance of the 3-SAT consists of $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, where each clause $C_i = \{x_i \vee y_i \vee z_i\}$ $(i = 1, \cdots, n)$, and $\{x_i, y_i, z_i\} \subset \{u_1, \bar{u}_1, \cdots, u_m, \bar{u}_m\}$. The decision problem is to determine whether we can assign a value (true or false) to each variable u_i, $i = 1, ..., m$, such that ϕ is true. We transform an instance of the 3-SAT problem to an instance of the mCK problem as follows. We consider a circle of diameter d'. Each variable u_i corresponds to a point on the circle, and we place its negation $\bar{u}_i$ diametrically opposite on the circle. Then the distance between u_i and $\bar{u}_i$ is d'. We set $d' = d + \epsilon$, where ϵ is a sufficiently small and positive value, such that the distance between any two points corresponding to different variables is no larger than d.

For each pair of variables u_i and $\bar{u}_i$, we create a keyword q_i $(i = 1, \cdots, m)$ and associate it with the points corresponding to u_i and $\bar{u}_i$. For each clause C_i, we create a keyword q_{m+i} $(i = 1, \cdots, n)$ and associate it with the points corresponding to the three variables in C_i. Therefore, given a 3-SAT instance ϕ, we have an mCK query q of $n + m$ keywords. If there exists a result of q of diameter at most d, then there exists a satisfying assignment for $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$. On the other hand, a satisfying assignment of the 3-SAT problem determines a set of points with diameter at most d covering all keywords in q. Therefore, the proof is complete. □

B. DETAILS OF *GKG*

The *GKG* algorithm is detailed in Algorithm 4. We first find the most infrequent query keyword t_{inf} (line 1). For each object o containing t_{inf}, we utilize the virtual bR*-tree indexing structure [22] to find the nearest object containing a term t to o. Alternatively, we can also use other geo-textual indexes in place (such as IR-tree [7]) and our algorithm *GKG* is equally applicable. We initialize a min priority queue $Queue$ to store the nodes and objects traversed, and their minimum distances to o are used as the key. Initially, the root node of the virtual bR*-tree is inserted into $Queue$ (line 6). If there are some keywords uncovered, we find the nearest object for each of them (lines 9–20). In each loop, we read the element e that is nearest to o (line 9). If it is an object, we insert it into $group$ and remove $e.\psi$ from $ucSet$ (lines 9–10). Otherwise, we read each of its child node e', and insert it into $Queue$ only if it covers some keywords in $ucSet$ (lines 14–20). We compare the group found around o with the current best group G_{gkg}, and update G_{gkg} if $group$ has smaller diameter (line 21).

C. PROOF OF THEOREM 4

PROOF. 1). If MCC_G is determined by two points in G, according to Theorem 3, the diameter of MCC_G is equal to the distance between the two points, and thus we have $\delta(G) = \emptyset(\mathsf{MCC}_G)$.

2). Consider that MCC_G is determined by three points A, B, and C in G. We use $\angle A$, $\angle B$, and $\angle C$ to denote the three angles of the triangle consisting of the three points. First, it is obvious that the distance between any two objects cannot exceed the diameter of the

Algorithm 4: *GKG* $(q, tree)$

```
1  G_gkg ← ∅;
2  t_inf ← the most infrequent keyword;
3  foreach object o containing t_inf do
4      group ← {o};
5      Queue ← new min-priority queue;
6      Queue.Enqueue(tree.root, 0);
7      ucSet ← q.ψ \ o.ψ ;            // uncovered keywords
8      while ucSet is not empty do
9          e ← Queue.Dequeue();
10         if e is an object then
11             group ← group ∪ e;
12             ucSet ← ucSet \ e.ψ;
13         else
14             foreach entry e' in node e do
15                 if ucSet ∩ e'.ψ ≠ ∅ then
16                     if e is a leaf node then
17                         dist ← Dist(e', q);
18                     else
19                         dist ← minDist(e', q);
20                     Queue.Enqueue(e', dist);
21     if G_gkg is ∅ or δ(group) < δ(G_gkg) then
22         G_gkg ← group;
23 return G_gkg;
```

Procedure findOSKEC(o, C_{cur})

```
1  olist ← a list of objects in O' whose distances to o are smaller than
     ø(C_cur), ranked by their distances to o;
2  foreach object o_j ∈ olist do
3      if Dist(o, o_j) > ø(C_cur) then
4          break;
5      foreach object o_m ∈ olist do
6          if Dist(o_m, o) ≥ Dist(o_j, o) or Dist(o_m, o_j) ≥ ø(C_cur)
           then
7              break;
8          if o_j = o_m then
9              C_can ← MCC_{o,o_j};
10         else
11             C_can ← MCC_{o,o_j,o_m};
12         if ø(C_can)<ø(C_cur) then
13             ucSet ← q.ψ \ o.ψ // uncovered keywords
14             foreach object o' ∈ olist do
15                 if o' in C_can then
16                     ucSet ← q.ψ \ o'.ψ;
17                 if ucSet is empty then
18                     C_cur ← C_can;
19                     break;
20 return C_cur;
```

circle, and thus we have $\delta(G) \leq \text{ø}(\mathsf{MCC}_G)$. Second, we assume $\angle C$ is the largest angle. Since $\angle A + \angle B + \angle C = 180^\circ$, we can conclude that $\angle C \geq 60^\circ$. According to Theorem 3, the triangle is not obtuse, and thus we know that $60^\circ \leq \angle C \leq 90^\circ$. Because MCC_G is also the circumcircle of the three points, we obtain that $\frac{\overline{AB}}{\sin \angle C} = \text{ø}(\mathsf{MCC}_G)$ according to the law of sines. Hence, we can get $\sin 60^\circ \leq \frac{\overline{AB}}{\text{ø}(\mathsf{MCC}_G)} \leq \sin 90^\circ$. Because $\delta(G)$ must be no smaller than $\overline{AB}$, we get $\frac{\sqrt{3}}{2}\text{ø}(\mathsf{MCC}_G) \leq \overline{AB} \leq \delta(G)$.

3). When there are more than three points on MCC_G, we can transform this to the cases that MCC_G is determined by either two or three points as follows: first, we select one point on MCC_G, and we remove it from G to check if the minimum covering circle of the remaining points is the same as MCC_G. If so, we ignore it; otherwise, we select the next point and repeat the above process. Finally, we will select two or three points that determines MCC_G, and then we can apply the proof in cases 1) and 2). □

D. PROCEDURE FINDOSKEC()

First, we read objects whose distances to o are smaller than the diameter of the current best circle C_{cur} (line 1). We process these objects in ascending order of their distances to o. When the second object o_j is fixed, we enumerate the third object o_m to obtain a candidate circle C_{can} (lines 7-10). Next, if C_{can} has a smaller diameter than the current best circle C_{cur}, we check if the objects in C_{can} can cover all query keywords (lines 11-18). If so, we update C_{cur} as C_{can} and begin to enumerate the next circle.

E. PROOF OF THEOREM 6

PROOF. We have $\delta(G_{skeca}) \leq \text{ø}(\mathsf{MCC}_{G_{skeca}})$ according to Theorem 4, and $\text{ø}(\mathsf{SKEC}_q) \leq \frac{2}{\sqrt{3}}\delta(G_{opt})$ according to Theorem 5. Hence, we can obtain $\delta(G_{skeca}) \leq \frac{2}{\sqrt{3}}\delta(G_{opt}) + \alpha = \frac{2}{\sqrt{3}}\delta(G_{opt}) + \epsilon\delta(G_{gkg})/2$. Because $\delta(G_{gkg}) \leq 2\delta(G_{opt})$, we have:

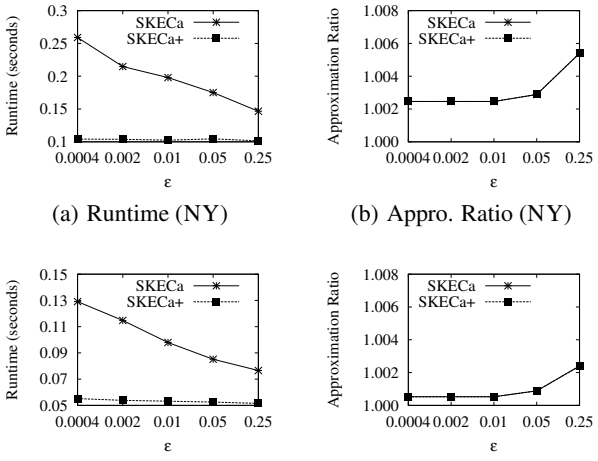
$$\frac{\delta(G_{skeca})}{\delta(G_{opt})} \leq \frac{2}{\sqrt{3}} + \epsilon\frac{\delta(G_{gkg})}{2\delta(G_{opt})} \leq \frac{2}{\sqrt{3}} + \epsilon.$$

Thus we complete the proof. □

F. TUNING THE BINARY SEARCH PARAMETER ϵ

We also study the effect of ϵ on the efficiency and accuracy of *SKECa* and *SKECa+* on datasets NY and TW. We vary ϵ from 0.0004 to 0.25. Figures 14(a) and 14(b) show the runtime and the accuracy of *SKECa* and *SKECa+* when we vary ϵ on the NY dataset, respectively. Figures 14(c) and 14(d) show the runtime and the accuracy of *SKECa* and *SKECa+* when we vary ϵ on the TW dataset, respectively.

It can be observed that the similar results are obtained to that on LA. *SKECa* always runs slower than *SKECa+*. As ϵ increases, their runtime drops, but the accuracy becomes worse. We observe that on NY and TW, setting ϵ to 0.01 can also balance the efficiency and accuracy well. Thus, we use 0.01 as the default value of ϵ for all experiments on all datasets.



(a) Runtime (NY) (b) Appro. Ratio (NY)

(c) Runtime (TW) (d) Appro. Ratio (TW)

Figure 14: Varying ϵ