Collective Spatial Keyword Queries: A Distance Owner-Driven Approach

Cheng Long[†] Raymond Chi-Wing Wong[†] Ke Wang[‡] Ada Wai-Chee Fu[§]
[†]The Hong Kong University of Science and Technology
[‡]Simon Fraser University
[§]The Chinese University of Hong Kong
[†]{clong,raywong}@cse.ust.hk [‡]wangk@cs.sfu.ca [§]adafu@cse.cuhk.edu.hk

ABSTRACT

Recently, spatial keyword queries become a hot topic in the literature. One example of these queries is the *collective spatial keyword query* (CoSKQ) which is to find a set of objects in the database such that it *covers* a set of given keywords collectively and has the smallest *cost*. Unfortunately, existing exact algorithms have severe scalability problems and existing approximate algorithms, though scalable, cannot guarantee near-to-optimal solutions. In this paper, we study the CoSKQ problem and address the above issues.

Firstly, we consider the CoSKQ problem using an existing cost measurement called the *maximum sum cost*. This problem is called MaxSum-CoSKQ and is known to be NP-hard. We observe that the maximum sum cost of a set of objects is dominated by at most *three* objects which we call the *distance owners* of the set. Motivated by this, we propose a distance owner-driven approach which involves two algorithms: one is an exact algorithm which runs faster than the best-known existing algorithm by several orders of magnitude and the other is an approximate algorithm which improves the best-known constant approximation factor from 2 to 1.375.

Secondly, we propose a new cost measurement called *diameter cost* and CoSKQ with this measurement is called Dia-CoSKQ. We prove that Dia-CoSKQ is NP-hard. With the same distance owner-driven approach, we design two algorithms for Dia-CoSKQ: one is an exact algorithm which is efficient and scalable and the other is an approximate algorithm which gives a $\sqrt{3}$-factor approximation.

We conducted extensive experiments on real datasets which verified that the proposed exact algorithms are scalable and the proposed approximate algorithms return near-to-optimal solutions.

Categories and Subject Descriptors

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

General Terms

Algorithms, Experimentation, Performance

Keywords

Spatial keyword querying, Distance owner-driven approach

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

1. INTRODUCTION

With the proliferation of spatial-textual data such as location-based services and geo-tagged websites, *spatial keyword queries* have been studied extensively recently [11, 8, 23, 3]. Given a set of spatial-textual objects and a query constituted by a location and a set of keywords, a typical spatial keyword query finds the object that *best* matches the arguments in the query. One example is to find the object closest to the query location among all objects that cover all the keywords specified in the query [23].

In some applications, users' needs (expressed as keywords) are satisfied by multiple objects *collectively* instead of a *single* object [4]. For instance, a tourist wants to have site-seeing, shopping and dining which could only be satisfied by *multiple* objects, e.g., tourist attractions, shopping malls and restaurants. Another example is that a user would like to set up a project consortium of partners within a certain region that combine to offer the capabilities required for the successful execution of the whole project. Finding multiple objects collectively to satisfy users' needs can be addressed by *Collective Spatial Keyword Query* (CoSKQ) [4].

Specifically, CoSKQ is described as follows. Let $\mathcal{O}$ be a set of objects. Each object $o \in \mathcal{O}$ is associated with a spatial location, denoted by $o.\lambda$, and a set of keywords, denoted by $o.\psi$. Given a query q with a location $q.\lambda$ and a set of keywords $q.\psi$, CoSKQ is to find a set S of objects such that S covers $q.\psi$, i.e., $q.\psi \subseteq \cup_{o \in S} o.\psi$, and the *cost* of S, denoted by $cost(S)$, is minimized.

There are different cost functions for $cost(S)$. One cost function is called the *maximum sum cost function*, denoted by $cost_{MaxSum}(S)$, and was studied in [4]. It is the linear combination of two *max* components: the maximum distance between q and an object in S and the maximum distance between two objects within S. CoSKQ adopting this cost function is called *MaxSum-CoSKQ*. The other cost function is called the *diameter cost function*, denoted by $cost_{Dia}(S)$. It is defined to be the *diameter* of $S \cup \{q\}$. In fact, diameter-related cost functions have been commonly adopted in graph databases [1, 13, 2, 15] and spatial databases [25, 26, 27]. To the best of our knowledge, we are the first to study this cost function for CoSKQ. CoSKQ adopting this cost function is called *Dia-CoSKQ*.

Given a query q, an object o is said to be *relevant* (to q) if o contains at least one keyword in $q.\psi$. We denote by $\mathcal{O}_q$ the set of all relevant objects to q. It is sufficient to focus on $\mathcal{O}_q$ only for a specific query q. Given a set S of objects, S is said to be *feasible* if S covers $q.\psi$. Thus, the optimal solution of CoSKQ is a feasible set with the smallest cost.

Although MaxSum-CoSKQ (which is proved to be NP-hard) has been studied by Cao et al. [4], the best-known exact algorithm which we call Cao-Exact is not scalable to large datasets and the two existing approximate algorithms which we call Cao-Appro1

and Cao-Appro2 do not have a very good theoretical guarantee. Specifically, Cao-Exact is a best-first search method based on the feasible set space whose size is $O(|\mathcal{O}_q|^{|q.\psi|})$. Though equipped with some pruning techniques, Cao-Exact is prohibitively expensive when the dataset is large. For example, in our experiments, Cao-Exact took more than 10 days for a query containing 6 keywords on a dataset with 8M objects.

In this paper, we propose two algorithms for MaxSum-CoSKQ, *MaxSum-Exact* and *MaxSum-Appro*. MaxSum-Exact is an exact algorithm and MaxSum-Appro is a 1.375-approximate algorithm.

MaxSum-Exact is more scalable compared with the best-known algorithm, Cao-Exact. A key observation which is used by MaxSum-Exact is that the number of distinct *costs* of all possible feasible sets is *cubic* (in terms of $|\mathcal{O}_q|$) although the number of all possible *feasible sets* is *exponential* (in terms of $|q.\psi|$). Given a feasible set S, the maximum sum cost function of S is dominated (or determined) by at most *three* objects in S, namely the object with the greatest distance from q and the two objects with the greatest pairwise distance within S. We say that these three objects form the *distance owner group* of S. Thus, the number of distinct costs of all possible feasible sets is bounded by the total number of all possible distance owner groups (which is bounded by $O(|\mathcal{O}_q|^3)$). Motivated by this, we propose a distance-owner driven approach called MaxSum-Exact for MaxSum-CoSK. MaxSum-Exact is a search algorithm based on the search space containing all possible distance owner groups. Besides, it incorporates some search strategies which can prune the search space effectively. Usually, *one* distance owner group corresponds to *many* feasible sets. This is verified by our experiments where MaxSum-Exact ran faster than Cao-Exact by 1-3 orders of magnitude.

MaxSum-Appro, the proposed approximate algorithm, improves the best-known constant approximation factor from 2 to 1.375 without incurring a higher worst-case time complexity.

Furthermore, we consider Dia-CoSKQ which has not been studied in the literature. In this paper, we prove that Dia-CoSKQ is NP-hard. We also adapt Cao-Exact, Cao-Appro1 and Cao-Appro2 for Dia-CoSKQ. However, these adapted algorithms suffer from the same drawbacks in MaxSum-CoSKQ.

Motivated by this, we propose two algorithms, namely *Dia-Exact* and *Dia-Appro*. Dia-Exact is an exact algorithm which is also a search algorithm based on the search space containing all possible *distance owner groups* and thus it is scalable to large datasets. Dia-Appro gives a $\sqrt{3}$-factor approximation for Dia-CoSKQ.

We summarize our main contributions as follows.

- Firstly, for MaxSum-CoSKQ, we design two algorithms, MaxSum-Exact and MaxSum-Appro. MaxSum-Exact is more scalable than the best-known exact algorithm, Cao-Exact. MaxSum-Appro improves the best-known constant approximation factor from 2 to 1.375 without incurring a higher worst-case time complexity.
- Secondly, for Dia-CoSKQ, which is new, we prove its NP-hardness and develop two algorithms, Dia-Exact and Dia-Appro. Dia-Exact significantly outperforms the adaptation of Cao-Exact, and Dia-Appro gives a $\sqrt{3}$-factor approximation.
- Thirdly, we conducted extensive experiments on both real and synthetic datasets, which verified our theoretical results and the efficiency of our algorithms.

The rest of this paper is organized as follows. Section 2 gives the definition of the CoSKQ problem and its existing solutions. Section 3 and Section 4 study MaxSum-CoSKQ and Dia-CoSKQ, respectively. Section 5 gives the empirical study and Section 6 reviews the related work. Section 7 concludes the paper.

2. BACKGROUND

2.1 Problem Definition

Let $\mathcal{O}$ be a set of objects. Each object $o \in \mathcal{O}$ is associated with a location denoted by $o.\lambda$ and a set of keywords denoted by $o.\psi$. Given two objects o and o', we denote by $d(o, o')$ the Euclidean distance between $o.\lambda$ and $o'.\lambda$. Given a query q which consists of a location $q.\lambda$ and a set of keywords $q.\psi$, we denote by $\mathcal{O}_q$ the set of **relevant** objects each of which contains at least one keyword in $q.\psi$, and say that a set of objects is **feasible** if it covers $q.\psi$. Besides, we introduce a fictitious object o_q in $\mathcal{O}$ with $o_q.\lambda = q.\lambda$ and $o_q.\psi = \emptyset$. For simplicity, we shall also refer to object o_q as q.

Problem Definition [4]. Given a query $q = (q.\lambda, q.\psi)$, the *Collective Spatial Keyword Query* (CoSKQ) problem is to find a set S of objects in $\mathcal{O}$ such that S covers $q.\psi$ and the *cost* of S is minimized.

In this paper, we consider two cost functions, the *maximum sum cost* and the *diameter cost*.

Given a set S of objects, the *maximum sum cost* of S, denoted by $cost_{MaxSum}(S)$, is equal to the linear combination of the maximum distance between q and an object in S and the maximum distance between two objects in S. That is,

$$cost_{MaxSum}(S) = \alpha \cdot \max_{o \in S} d(o, q) + (1 - \alpha) \cdot \max_{o_1, o_2 \in S} d(o_1, o_2) \quad (1)$$

where $\alpha \in [0, 1]$ is a user parameter. Same as [4], for ease of exposition, we consider the case where $\alpha = 0.5$ only. In this case, we can safely assume that

$$cost_{MaxSum}(S) = \max_{o \in S} d(o, q) + \max_{o_1, o_2 \in S} d(o_1, o_2) \quad (2)$$

In fact, the applicability of all of our algorithms does not rely on the setting of α. The only part that is affected is the approximation factor of our approximate algorithm which is *bounded* by $(2 - \alpha)$ (e.g., when $\alpha = 0.5$, the approximation factor of our approximate algorithm is 1.375 which is bounded by $(2 - \alpha) = 1.5$). More discussion on the general case of α could be found in [16]. The CoSKQ problem using this cost is called *MaxSum-CoSKQ*.

As could be noticed, parameter α in the maximum sum cost function is used to balance the two *max* components, namely $\max_{o \in S} d(o, q)$ and $\max_{o_1, o_2 \in S} d(o_1, o_2)$. Sometimes, however, people may not have a concrete idea of how to specify α. To ease this situation, we define an alternative cost function called *diameter cost* on a set S of objects, denoted by $cost_{Dia}(S)$, which is defined to be the larger of these two *max* components. That is,

$$cost_{Dia}(S) = \max_{o_1, o_2 \in S \cup \{o_q\}} d(o_1, o_2) \quad (3)$$

The CoSKQ problem using this cost is called *Dia-CoSKQ*.

Intractability. It has been proved in [4] that MaxSum-CoSKQ is NP-hard. In this paper, we prove that Dia-CoSKQ is also NP-hard.

LEMMA 1. *Dia-CoSKQ is NP-hard.* □

PROOF. For interest of space, our proof can be found in the full version of this paper [16]. We can show this by transforming an existing NP-complete problem, 3-SAT, to Dia-CoSKQ. □

2.2 Existing Solutions for MaxSum-CoSKQ

Cao et el. [4] proposed one exact algorithm, Cao-Exact, and two approximate algorithms, Cao-Appro1 and Cao-Appro2, for MaxSum-CoSKQ.

Cao-Exact. Cao-Exact is a best-first search method using an index called *IR-tree* [8]. An IR-tree is an R-tree in which each node is augmented with an *Inverted File* (IF). Consider a leaf node N. For

each keyword t, we construct an *inverted list* which is a list of all objects in node N containing t. All inverted lists in this leaf node N form the IF of N. Consider a non-leaf node N'. For each keyword t, we construct an *inverted list* which is a list of all child nodes in N' *covering* t. Given a keyword t, a node N'' is said to cover t if there exists an object in the subtree rooted at N'' containing t. All inverted lists in this non-leaf node N' form the IF of N'.

Cao-Exact is basically an exhaustive search on the object space with some pruning strategies in the IR-tree. The worst-case time complexity of Cao-Exact is $O(|\mathcal{O}|^{|q.\psi|})$, which corresponds to the size of the set containing all possible feasible sets.

Cao-Appro1. Cao-Appro1 gives a 3-factor approximation for MaxSum-CoSKQ. Specifically, Cao-Appro1 finds for each $t \in q.\psi$, q's nearest neighbor (NN) in $\mathcal{O}$ containing t and returns the set containing all these NNs as the approximate solution. Since Cao-Appro1 issues NN queries at most $|q.\psi|$ times and each NN query takes $O(\log |\mathcal{O}|)$ time [6, 10, 18], the time complexity of Cao-Appro1 is $O(|q.\psi| \cdot \log |\mathcal{O}|)$.

Cao-Appro2. Cao-Appro2 gives a 2-factor approximation for MaxSum-CoSKQ. Specifically, Cao-Appro2 enhances Cao-Appro1 as follows. First, Cao-Appro2 invokes Cao-Appro1 and obtains an approximate solution denoted by S_1. Let o_f be the farthest object from q in S_1 and t_f be a keyword contained by o_f but not contained by any other *closer* object from q in $\mathcal{O}$. Then, for each object o in $\mathcal{O}$ containing t_f, it finds for each keyword t in $q.\psi$, o's nearest object that contains t in $\mathcal{O}$ and obtains a corresponding approximate solution containing all these NNs. Among all these approximate solutions as well as S_1, it returns the one with the smallest cost. Thus, the approximate solution returned by Cao-Appro2 is no worse than that returned by Cao-Appro1. Since there are at most $|\mathcal{O}_q|$ objects containing t_f and the cost for each such object is simply $O(|q.\psi| \cdot \log |\mathcal{O}|)$, the worst-case time complexity of Cao-Appro2 is $O(|\mathcal{O}_q| \cdot |q.\psi| \cdot \log |\mathcal{O}|)$.

3. ALGORITHMS FOR MAXSUM-COSKQ

In this section, we propose two algorithms, MaxSum-Exact (Section 3.1) and MaxSum-Appro (Section 3.2), for MaxSum-CoSKQ. For clarity, we simply write $cost_{MaxSum}(\cdot)$ as $cost(\cdot)$ if the context of the cost function is clear.

Given a query q and a non-negative real number r, we denote the *circle* or the *disk* centered at $q.\lambda$ with radius r by $D(q, r)$. Given a disk D, we denote the radius of D by $radius(D)$. Given a query q, a disk centered at $q.\lambda$ is called a *q-disk*. Given a q-disk D and an object o in D, o is said to be the *boundary object* of D if there does not exist other objects o' in D such that $d(o', q) > d(o, q)$. Note that in some cases, a boundary object of a disk is along the boundary of a disk and in some other cases, it is inside the disk without touching the boundary of the disk.

3.1 Finding Optimal Solution

In this section, we propose an exact algorithm called *MaxSum-Exact*. The key to the efficiency of MaxSum-Exact is based on the splitting property of the maximum sum cost function.

3.1.1 Splitting Property

Let S' be a feasible set. The maximum sum cost of S' can be split into two parts, namely the *query distance cost* which is $\max_{o \in S'} d(o, q)$ and the *pairwise distance cost* which is $\max_{o_1, o_2 \in S'} d(o_1, o_2)$. We define the **query distance owner** of S' to be o where $o = \arg\max_{o \in S'} d(o, q)$. We also define the **pairwise distance owners** of S' to be o_1 and o_2 where $(o_1, o_2) = \arg\max_{(o'_1, o'_2) \in S' \times S'} d(o'_1, o'_2)$.

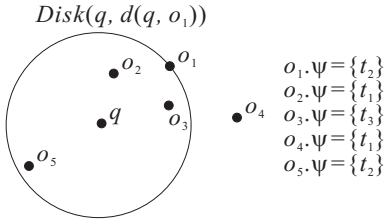


Figure 1: An example

Consider Figure 1 containing a query location q and 5 objects, namely o_1, o_2, o_3, o_4 and o_5. The set of keywords associated with each object can be found in the figure. Suppose that $q.\psi = \{t_1, t_2, t_3\}$. We know that a set $S' = \{o_1, o_2, o_3\}$ is feasible. The query distance owner of S' is o_1 and the pairwise distance owners of S' are o_2 and o_3.

According to the above splitting property, the cost of a set S' can be dominated (or determined) by exactly three objects in S', namely the query distance owner of S' (i.e., o) and the two pairwise distance owners of S' (i.e., o_1 and o_2). In other words, we can simply write the cost of S' as follows.

$$cost(S') = d(o, q) + d(o_1, o_2)$$

where o is the distance owner of S', and o_1 and o_2 are the two pairwise distance owners of S'. We say that o, o_1 and o_2 forms a **distance owner group**. Any feasible set with its query distance owner as o and its pairwise distance owners as o_1 and o_2 is said to be (o, o_1, o_2)**-owner consistent**. Note that each feasible set that is (o, o_1, o_2)-owner consistent has the same cost equal to $d(o, q) + d(o_1, o_2)$.

3.1.2 Distance Owner-Driven Approach

Based on the splitting property, we propose a *distance owner-driven approach* as follows. This approach maintains a variable S storing the best feasible set found so far. Initially, S is set to a feasible set (We will describe how we find this feasible set later). Then, it has four major steps.

- *Step 1 (Query Distance Owner Finding):* Select one object o in $\mathcal{O}_q$ to take the role of the query distance owner of a set S' *to be found.*
- *Step 2 (Pairwise Distance Owner Finding):* Select two objects, o_1 and o_2, in $\mathcal{O}_q$ to take the roles of the pairwise distance owners of the set S' (to be found). Note that o, o_1 and o_2 form a distance owner group.
- *Step 3 (Sub-Optimal Feasible Set Finding):* Find the set S' which is (o, o_1, o_2)-owner consistent (if any), and update S with S' if $cost(S') < cost(S)$.
- *Step 4 (Iterative Step):* Repeat Step 1 and Step 2 which find another *distance owner group*, and continue with Step 3 until all distance owner groups are traversed.

The above approach gives a search strategy based on the set of all possible distance owner groups. However, a straightforward implementation of this approach would enumerate all $|\mathcal{O}_q|^3$ distance owner groups, which is prohibitively expensive in practice. Thus, we need a careful design in order to prune the search space effectively. In the following, we elaborate the pruning features enjoyed by this distance owner-driven approach, which cannot be found in the best-known algorithm, Cao-Exact.

Firstly, some objects in $\mathcal{O}_q$ need not be considered in Step 2 after we select an object in Step 1. To illustrate this, consider Figure 1. Suppose that we pick o_1 as the query distance owner in Step 1. We do not need to consider o_4 as objects in Step 2. This is because $d(o_4, q)$ is larger than $d(o_1, q)$, which violates the property that

o_1 takes the role of the query distance owner of the set S' to be found if S' contains o_1 and o_4. We formalize this pruning feature as follows.

PROPERTY 1 (PRUNING). *Let S' be a feasible set. If o is the query distance owner of S', then the two pairwise distance owners of S' are inside $D(q, d(o, q))$.* □

PROOF. Any object $o' \in S'$ has $d(o', q) \leq d(o, q)$ and thus o' is inside $D(q, d(o, q))$. □

Secondly, most of the objects in $\mathcal{O}_q$ need not be considered to form a set S' to be found in Step 3. To illustrate this, consider Figure 1 again. Suppose that we pick o_1 as the query distance owner in Step 1, and o_2 and o_3 as the pairwise distance owners in Step 2. Similarly, we still do not need to consider o_4 as one of the objects to form the set S' since including o_4 violates the query distance owner property. Besides, we do not need to consider o_5 to form the set S' to be found. This is because $d(o_2, o_5) > d(o_2, o_3)$ which violates the property that o_2 and o_3 take the roles of the pairwise distance owners. Similarly, we formalize this pruning feature as follows.

PROPERTY 2 (PRUNING). *Let S' be a feasible set. If o is the query distance owner of S', and o_1 and o_2 are two pairwise distance owners of S', then all objects in S' are inside $\mathcal{R}$ where $\mathcal{R} = D(q, d(o, q)) \cap D(o_1, d(o_1, o_2)) \cap D(o_2, d(o_1, o_2))$.* □

PROOF. For each $o' \in S'$, we have $d(o', q) \leq d(o, q)$ which implies that o' is inside $D(q, d(o, q))$. For each $o' \in S'$, we have $d(o', o_1) \leq d(o_1, o_2)$ which implies that o' is inside $D(o_1, d(o_1, o_2))$, and $d(o', o_2) \leq d(o_1, o_2)$ which implies that o' is inside $D(o_2, d(o_1, o_2))$. □

The above pruning features look promising for improving the efficiency of the proposed approach. Moreover, since objects *near* to q usually form the optimal set, we propose to consider the objects in Step 1 iteratively, taking the role of the query distance owner of the set to be found, in ascending order of their distances to q in order to further improve the efficiency of the proposed approach.

Usually, the NN of q in $\mathcal{O}_q$ is not the query distance owner of the set S' to be found. In Figure 1, consider the query q with its keyword set to be $\{t_1, t_2, t_3\}$. The NN of q is o_2. Suppose that o_2 is the query distance owner of S'. According to Property 2, all objects in S' fall in $D(q, d(o_2, q))$ and they together cover $q.\psi$. But, in the figure, no object in $D(q, d(o_2, q))$ contains t_2, which implies that we cannot find a feasible set S' with o_2 as its query distance owner.

Based on this observation, we propose to find the *closest possible query distance owner*, say o, of the set S' to be found such that there exists a feasible set in the q-disk $D(q, d(o, q))$. In addition, we do not want to pick any object which is far away from q. Thus, we also propose to find the *farthest possible query distance owner* of S' to be found that we need to consider.

3.1.3 Closest/Farthest Possible Query Dist. Owner

The following two lemmas show how to find the closest and farthest possible query distance owners.

Before we present the first lemma about the closest possible query distance owner, we introduce some notations. Given a query q and a keyword t, the **t-keyword nearest neighbor** of q, denoted by $NN(q, t)$, is defined to be the NN of q containing keyword t. We have a similar definition on $NN(o, t)$ for an object o. We define the **nearest neighbor set** of q, denoted by $N(q)$, to be the set containing q's t-keyword nearest neighbor for each $t \in q.\psi$, i.e., $N(q)$ is $\cup_{t \in q.\psi} NN(q, t)$. Note that $N(q)$ is a feasible set.

LEMMA 2 (CLOSEST POSS. QUERY DIST. OWNER). *Let $r_{min} = \max_{o \in N(q)} d(o, q)$. There exists a feasible set in a q-disk D if and only if $radius(D) \geq r_{min}$.* □

PROOF. The proof for the "if" part is trivial since for any q-disk D with $radius(D) \geq r_{min}$, $N(q)$ is a feasible set in D. We prove the "only if" part by contradiction. Assume $radius(D) < r_{min}$ and there exists a feasible set S in D. Let o_f be the farthest object from q in $N(q)$, i.e., $r_{min} = d(q, o_f)$. There exists a keyword $t_f \in o_f.\psi \cap q.\psi$ such that t_f is not contained by any object that is closer to q than o_f since otherwise $o_f \notin N(q)$. Since S is feasible, there exists an object $o \in S$ that contains keyword t_f. As a result, we have $d(o, q) \leq radius(D) < r_{min} = d(q, o_f)$, which leads to a contradiction. □

The above lemma suggests that there is no feasible set in a q-disk D if $radius(D) < r_{min}$. Thus, the disk with its radius equal to r_{min} is the "smallest" disk we need to consider. The boundary object of this disk is the closest possible query distance owner. Note that this object is along the boundary of this disk.

The following lemma gives the "largest" disk we need to consider. Besides, the boundary object of this disk corresponds to the farthest possible query distance owner. Note that this object might or might not be along the boundary of this disk.

LEMMA 3 (FARTHEST POSS. QUERY DIST. OWNER). *Let S be a feasible set and $r_{max} = cost(S)$. Let D be a q-disk with $radius(D) > r_{max}$. Then, for any feasible set S' containing at least one object outside D, $cost(S') > cost(S)$.* □

PROOF. $cost(S') \geq max_{o \in S'} d(o, q) > radius(D) > r_{max} = cost(S)$. □

The above lemma suggests that when we have known a feasible set S, there is no need to consider the objects outside $D(q, r_{max})$ where $r_{max} = cost(S)$.

The above two lemmas suggest the "smallest" disk and the "largest" disk we need to consider. Specifically, the object o which takes the role of the query distance owner of S' to be found must be in the *ring* which is roughly equal to the "largest" disk minus the "smallest" disk. Let S be a feasible set. Let $r_{min} = \max_{o \in N(q)} d(o, q)$ and $r_{max} = cost(S)$. We define the **ring** for S, denoted by $R(S)$, to be $D(q, r_{max}) - D(q, r_{min} - \delta)$, where δ is a very small positive real number near to 0.

LEMMA 4 (RING CANDIDATE). *Let S be a feasible set and S_o be the optimal set for the MaxSum-CoSKQ problem. The query distance owner of S_o is inside $R(S)$.* □

PROOF. Let o be the query distance owner of S_o. First, according to Lemma 3, o cannot be outside $D(q, r_{max})$ since otherwise $cost(S_o) > cost(S)$ which leads to a contradiction. Second, according to Lemma 2, there exist no feasible sets in $D(q, r_{min} - \delta)$. Thus, o is not inside $D(q, r_{min} - \delta)$ since otherwise S_o which is feasible is inside $D(q, r_{min} - \delta)$ which also leads to a contradiction. Therefore, o is inside $R(S)$. □

It is easy to verify that the region occupied by $R(S)$ becomes smaller when $cost(S)$ is smaller since the radius of the outer disk of $R(S)$ is equal to $cost(S)$.

3.1.4 The MaxSum-Exact Algorithm

Based on the discussion in the previous subsection, we design *MaxSum-Exact* as shown in Algorithm 1. Specifically, we maintain S for storing the best-known solution found so far, which is

Algorithm 1 Algorithm *MaxSum-Exact*

Input: query q and a set $\mathcal{O}$ of objects

1: $S \leftarrow N(q)$
2: **while** there is an "un-processed" relevant object o in $R(S)$ **do**
3: // Step 1 (Query Distance Owner Finding)
4: $o \leftarrow$ the nearest "un-processed" relevant object in $R(S)$
5: // Step 2 (Pairwise Distance Owner Finding)
6: $D \leftarrow$ the q-disk with its radius equal to $d(o, q)$
7: $P \leftarrow$ a set of all pairs (o_1, o_2) where o_1 and o_2 are in D
8: // Step 3 (Sub-optimal Feasible Set Finding)
9: **for** each $(o_1, o_2) \in P$ in ascending order of $d(o_1, o_2)$ **do**
10: **if** there exists a feasible set S' in D which is (o, o_1, o_2)-owner consistent **then**
11: **if** $cost(S') < cost(S)$ **then**
12: $S \leftarrow S'$; **break**
13: // Step 4 (Iterative Process)
14: mark o as "processed"
15: **return** S

initialized to $N(q)$. Then, we perform an iterative process as follows. Consider an iteration. We want to check whether there exists a relevant object in $R(S)$ that has not been processed. If yes, we pick the nearest relevant object o from $R(S)$ that has not been processed to take the role of the query distance owner of the set S' to be found (Step 1). This object is said to be the *query distance owner* for this iteration. We process it as follows. Firstly, we form the q-disk D with its radius equal to $d(o, q)$ and find a set P of all pairs (o_1, o_2) where o_1 and o_2 are in D for taking the roles of the pairwise distance owners (Step 2). Secondly, for each pair (o_1, o_2) in P which is processed in ascending order of $d(o_1, o_2)$, we check whether there exists a feasible set S' which is (o, o_1, o_2)-owner consistent. Case 1: yes. We do the following. Firstly, if $cost(S') < cost(S)$, then we update S by S'. Secondly, we terminate to search the remaining pairs in P since the cost of a final set whose pairwise distance owners corresponds to one of the remaining pairs must be at least the cost of the current set S' whose pairwise distance owners are (o_1, o_2), the current processed pair. Case 2: no. We continue to consider the next pair in P until Case 1 is reached or all the pairs in P have been processed. We continue the above iteration with the next relevant object from $R(S)$ that has not been processed until all objects in $R(S)$ have been processed (Step 4).

We verify the correctness of MaxSum-Exact via Theorem 1.

THEOREM 1. *MaxSum-Exact returns a feasible set with the smallest cost for MaxSum-CoSKQ.* □

PROOF. Let S_o be one of the feasible sets with the smallest cost for MaxSum-CoSKQ. Suppose that o is the query distance owner of S_o, and o_1 and o_2 are two pairwise distance owners of S_o. According to Lemma 4, o is inside $R(S)$, where S is the solution maintained in MaxSum-Exact. Thus, o must have been processed in MaxSum-Exact (Step 1). When o is processed, pair (o_1, o_2) is included in P (Step 2) since o_1 and o_2 are inside $D(q, d(o, q))$ (Property 1). As a result, any feasible set which is (o, o_1, o_2)-owner consistent is retrieved (Step 3) and used to update S (there must exist some since S_o is (o, o_1, o_2)-owner consistent). The resulting S will not be updated anymore since it has the same cost as $cost(S_o)$ which is the smallest, and thus S is the final output. □

Algorithm 1 looks straightforward but how to execute this algorithm *efficiently* needs more careful design. We propose two computation strategies in the algorithm, namely the *self-iteration computation strategy* and the *cross-iteration computation strategy*, to execute this algorithm efficiently. The self-iteration computation strategy is to speed up the operations within an iteration and the cross-iteration computation strategy is to speed up the operations across different iterations.

Self-Iteration Computation Strategy. Consider an iteration in the algorithm whose query distance owner is o. Step 1 (lines 3-4) is straightforward. In Step 2 (lines 5-7), there is a step of finding a set P of all pairs (o_1, o_2) where o_1 and o_2 are in D. There is no need to keep all pairs (o_1, o_2) in P and some pairs can be pruned. The following two lemmas give some hints for pruning. The first lemma (Lemma 5) is based on the triangle inequality and the second lemma (Lemma 6) is based on the best-known set S found so far.

LEMMA 5 (TRIANGLE INEQUALITY). *Let S' be a feasible solution whose query distance owner is o, and pairwise distance owners are o_1 and o_2. Then, $d(o_1, o_2) \geq d(o, q) - \min\{d(o_1, q), d(o_2, q)\}$.* □

PROOF. Note that $d(o_1, o_2) \geq d(o_1, o)$ and $d(o_1, o_2) \geq d(o_2, o)$. By the triangle inequality, we know $d(o_1, o) \geq d(o, q) - d(o_1, q)$ and $d(o_2, o) \geq d(o, q) - d(o_2, q)$. Thus, we have $d(o_1, o_2) \geq d(o, q) - \min\{d(o_1, q), d(o_2, q)\}$. □

The above lemma suggests that the pair (o_1, o_2) in P can be pruned if $d(o_1, o_2) < d(o, q) - \min\{d(o_1, q), d(o_2, q)\}$. Let $d_{min} = d(o, q) - \min\{d(o_1, q), d(o_2, q)\}$. Thus, d_{min} corresponds to the smallest distance threshold for a pair (o_1, o_2).

LEMMA 6 (BEST KNOWN SET). *Let S' be a feasible solution whose query distance owner is o and pairwise distance owners are o_1 and o_2. Let S be another feasible solution. $cost(S') \leq cost(S)$ if and only if $d(o_1, o_2) \leq cost(S) - d(o, q)$.* □

PROOF. $cost(S') \leq cost(S)$ deduces $d(o, q) + d(o_1, o_2) \leq cost(S)$ which is exactly $d(o, q) \leq cost(S) - d(o_1, o_2)$. □

Let S be the feasible set found so far in the algorithm. The above lemma suggests that the pair (o_1, o_2) in P can be pruned if $d(o_1, o_2) > cost(S) - d(o, q)$. Let $d_{max} = cost(S) - d(o, q)$. Thus, d_{max} is the largest distance threshold for a pair (o_1, o_2).

According to Lemma 5 and Lemma 6, we only need to maintain those pairs with their distances between d_{min} and d_{max} in P.

Consider Step 3 (lines 8-12). Here, we need to process each pair (o_1, o_2) in P. The most time-consuming operation is to check whether there exists a feasible set S' which is (o, o_1, o_2)-owner consistent. Algorithm 2 presents an algorithm for this task. If it succeeds, it outputs S'; otherwise, it outputs $\emptyset$. First, it checks whether $d(o_1, o_2) < \max\{d(o_1, o), d(o_2, o)\}$. If yes, we conclude that there exist no feasible set that is (o, o_1, o_2)-owner consistent since it violates the condition that o_1 and o_2 are the pairwise distance owners (i.e., $d(o_1, o_2) \geq \max\{d(o_1, o), d(o_2, o)\}$). If no, it initializes S' to be $\{o, o_1, o_2\}$. It also maintains a variable ψ, denoting the set of keywords not covered by S' yet, which is initialized as $q.\psi - (o.\psi \cup o_1.\psi \cup o_2.\psi)$. If $\psi = \emptyset$, it returns S' immediately. Otherwise, it proceeds to augment S' with some other objects. According to Property 2, we can safely focus on the region $\mathcal{R} = D(o, d(o, q)) \cap D(o_1, d(o_1, o_2)) \cap D(o_2, d(o_1, o_2))$. Therefore, it retrieves the set $\mathcal{O}'$ of all relevant objects in $\mathcal{R}$. If $\mathcal{O}'$ does not cover ψ, it returns $\emptyset$. Otherwise, it enumerates each possible subset S'' of $\mathcal{O}'$ that covers ψ (by utilizing the *inverted lists* maintained for each keyword in ψ), augment S' by S'' (thus S' becomes feasible) and checks whether S' is (o, o_1, o_2)-owner consistent which is equivalent to checking whether o_1 and o_2 are still

Algorithm 2 Algorithm for checking whether there exists a feasible set S' which is (o, o_1, o_2)-owner consistent

Input: three objects o, o_1 and o_2
Output: a feasible set which is (o, o_1, o_2)-owner consistent if any and $\emptyset$ otherwise

1: **if** $d(o_1, o_2) < \max\{d(o_1, o), d(o_2, o)\}$ **then return** $\emptyset$
2: $S' \leftarrow \{o, o_1, o_2\}$
3: $\psi \leftarrow q.\psi - (o.\psi \cup o_1.\psi \cup o_2.\psi)$
4: **if** $\psi = \emptyset$ **then return** S'
5: $\mathcal{R} \leftarrow D(q, d(o, q)) \cap D(o_1, d(o_1, o_2)) \cap D(o_2, d(o_1, o_2))$
6: $\mathcal{O}' \leftarrow$ a set of all relevant objects in $\mathcal{R}$
7: **if** $\mathcal{O}'$ does not cover ψ **then return** $\emptyset$
8: **for** each subset S'' of $\mathcal{O}'$ that covers ψ **do**
9: $\quad S' \leftarrow S' \cup S''$
10: $\quad$ **if** S' is (o, o_1, o_2)-owner consistent **then return** S'
11: $\quad S' \leftarrow S' - S''$
12: **return** $\emptyset$

the pairwise distance owners of S'. If yes, it outputs S'. Otherwise, it restores S' and checks the next subset of $\mathcal{O}'$. When all subsets of $\mathcal{O}'$ that cover ψ have been traversed and still no feasible set S' which is (o, o_1, o_2)-owner consistent has been found, it returns $\emptyset$.

Cross-Iteration Computation Strategy. We reuse the information computed in the previous iterations for the current iteration.

Consider an iteration where the query distance owner for this iteration is o. With respect to o, we create a q-disk D and also construct set P (line 7 in Algorithm 1). Consider the next iteration where the query distance owner for this iteration is o'. Although we can construct set P' with respect to o' from scratch by applying the procedure of generating set P, a much better approach is to construct set P' by using the current content of P because $P \subseteq P'$. Specifically, when we consider the next iteration, we first construct another set Q to be the set of additional pairs in P' compared with P (i.e., $Q = \{(o'', o') | o'' \in D(q, d(q, o))\}$) and then set P' to be $P \cup Q$. Note that $P \cap Q = \emptyset$.

The pruning in P mentioned in the self-iteration computation strategy is still valid even when we construct P' in the above way. Specifically, the pairs pruned previously in P still do not need to be considered in P' at the next iteration. This is because d_{min} is *monotonically increasing* and d_{max} is *monotonically decreasing* with more iterations. To illustrate, consider a pair (o_1, o_2) in P at the previous iteration. Note that $d_{min} = d(o, q) - \min\{d(o_1, q), d(o_2, q)\}$ and $d_{max} = cost(S) - d(o, q)$. At the next iteration, o will become o', which is at least as far as o from q (i.e., $d(o', q) \geq d(o, q)$). Thus, at the next iteration, d_{min} will remain the same or will increase. In addition, the cost of the solution S maintained at the next iteration is at most the cost of that maintained at the previous iteration. Thus, at the next iteration, d_{max} will remain the same or will decrease.

3.1.5 Implementation and Time Complexity

We adopt the IR-tree built on $\mathcal{O}$ to support both the NN query (line 1 of Algorithm 1) and the range query (line 7 of Algorithm 1 and line 6 of Algorithm 2). For the NN query, we adopt the best-first search method [12] and for the range query, we perform a simple breadth-first traversal with the constraint of the range. Besides, given a query q, since we only focus on the set of relevant objects, when performing NN queries and range queries, we can utilize the IF information maintained in the IR-tree for pruning.

Since the pairs in P are processed in ascending order of their distances and P is maintained dynamically (because of the Cross-Iteration Computation Strategy), we adopt a *binary search tree* for maintaining P, which allows efficient sorting and update.

Let n_1 be the number of iterations (lines 2-14) in MaxSum-Exact (Algorithm 1). Note that $n_1 << |\mathcal{O}_q|$ since n_1 corresponds to the number of relevant objects we process in $R(S)$ and the area occupied by $R(S)$ is typically small. Let $|P|$ be the size of the set P we use in the algorithm. Similarly, we know that $|P| << |\mathcal{O}_q|^2$. Let β be the cost of Algorithm 2. It is easy to verify that the time complexity of MaxSum-Exact is $O(n_1 \cdot |P| \cdot \beta)$.

Next, we analyze β. The cost of lines 1-4 (Algorithm 2) is dominated by those of other parts in the algorithm. The cost of lines 5-6 is simply $O(\log |\mathcal{O}| + |\mathcal{O}_q|)$ since we can issue three range queries and then perform an intersection on the query results. The cost of line 7 is $O(|\psi| \cdot |\mathcal{O}_q|)$. The cost of lines 8-11 is $O(|\mathcal{O}'|^{|\psi|} \cdot |\psi|^2)$ since it enumerates at most $O(|\mathcal{O}'|^{|\psi|})$ subsets S'' that cover ψ and each subset incurs a checking operation (line 10) whose cost is $O(|\psi|^2)$ (since $|S''| = O(|\psi|)$ and we can try all pairwise distances within S'' to do the checking). Thus, β is $O(\log |\mathcal{O}| + |\mathcal{O}_q| + |\psi| \cdot |\mathcal{O}_q| + |\mathcal{O}'|^{|\psi|} \cdot |\psi|^2)$. Note that $|\mathcal{O}'| << |\mathcal{O}_q|$ (since $\mathcal{O}'$ corresponds to a set of relevant objects in a small region), $|\mathcal{O}_q| < |\mathcal{O}|$ and $|\psi| \leq |q.\psi| - 1$.

In conclusion, the time complexity of MaxSum-Exact is $O(n_1 \cdot |P| \cdot (\log |\mathcal{O}| + |\mathcal{O}_q| + |\psi| \cdot |\mathcal{O}_q| + |\mathcal{O}'|^{|\psi|} \cdot |\psi|^2))$.

3.2 Finding Approximate Solution

In this section, we propose a 1.375-factor approximate algorithm called *MaxSum-Appro* which is better than the best-known 2-factor approximate algorithm, Cao-Appro2.

Before we present MaxSum-Appro, we introduce the concept of "o-neighborhood feasible set". Given a query q and an object $o \in \mathcal{O}$, the **o-neighborhood feasible set** is defined to be the set containing o and all other objects each of which is the t-keyword nearest neighbor of o in $D(q, d(o, q))$ for each $t \in q.\psi - o.\psi$. For example, consider Figure 1. Suppose that the query $q.\psi$ is $\{t_1, t_2, t_3\}$. Then, the o_1-neighborhood feasible set is $\{o_1, o_2, o_3\}$ since $q.\psi - o_1.\psi = \{t_1, t_3\}$, o_1's t_1-keyword nearest neighbor in $D(q, d(o_1, q))$ is o_2 and o_1's t_3-keyword nearest neighbor in $D(q, d(o_1, q))$ is o_3. It could be easily verified that the o-neighborhood feasible set exists iff o is outside $D(q, r_{min} - \delta)$ since an o-neighborhood feasible set is a feasible set.

In MaxSum-Appro, we only consider the o-neighborhood feasible sets for those objects o that are inside $R(S)$ where S is a feasible set, and thus they always exist.

We present MaxSum-Appro in Algorithm 3. MaxSum-Appro is exactly Algorithm 1 by replacing Step 2 and Step 3 which are relatively expensive with the new efficient operation of finding the o-neighborhood feasible set which could be finished by issuing $|q.\psi - o.\psi|$ NN queries.

Theoretical Analysis. Although the set S returned by the MaxSum-Appro algorithm might have a larger cost than the optimal set S_o, the difference is bounded.

THEOREM 2. *MaxSum-Appro gives a 1.375-factor approximation for the MaxSum-CoSKQ problem.* □

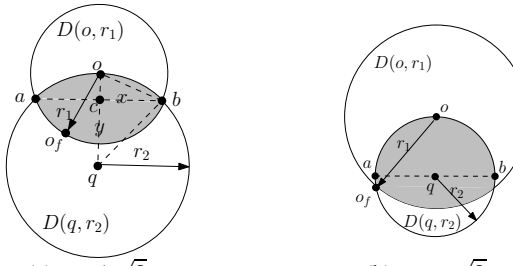
PROOF. Let S_o be the optimal solution and S be the solution returned by MaxSum-Appro. Let o be the query distance owner of S_o. By Lemma 4, we know that o is in $R(S)$. Besides, we can safely assume that o is a relevant object. Thus, there exists an iteration in MaxSum-Appro such that we process o (line 3) and thus we find its o-neighborhood feasible set denoted by S'.

Since S is the final solution returned by MaxSum-Appro, we know that $cost(S) \leq cost(S')$. The remaining part of the proof shows that $cost(S') \leq 1.375 \cdot cost(S_o)$.

Algorithm 3 Algorithm *MaxSum-Appro*

Input: query q and a set $\mathcal{O}$ of objects

1: $S \leftarrow N(q)$
2: **while** there is an "un-processed" relevant object o in $R(S)$ **do**
3: // Step 1 (Query Distance Owner Finding)
4: $o \leftarrow$ the nearest "un-processed" relevant object in $R(S)$
5: // Step 2 (o-Neighborhood Feasible Set Finding)
6: $S' \leftarrow$ the o-neighborhood feasible set
7: **if** $cost(S') < cost(S)$ **then**
8: $S \leftarrow S'$
9: // Step 3 (Iterative Process)
10: mark o as "processed"
11: **return** S



(a) $r_1 \leq \sqrt{2}r_2$ (b) $r_1 > \sqrt{2}r_2$

Figure 2: Illustration of the proof of Theorem 2

Let o_f be the object in S' that is the farthest from o and $r_1 = d(o_f, o)$. Then, all objects in S' fall in $D(o, r_1)$. Let $r_2 = d(o, q)$. Since o is the query distance owner of S', we know that all objects in S' fall in $D(q, r_2)$. In summary, all objects in S' fall in $D(o, r_1) \cap D(q, r_2)$.

Consider $cost(S_o)$. It could be verified by using a similar method for proving Lemma 2 that $\max_{o_1,o_2 \in S_o} d(o_1, o_2) \geq d(o, o_f)$. Thus, we have $cost(S_o) \geq r_2 + r_1$.

In the following, we consider two cases on r_1 according to whether there exists a line segment linking two points at the boundary of $D(q, r_2)$ such that it has its length equal to $2r_2$ (i.e., the diameter of $D(q, r_2)$) and falls in $D(o, r_1) \cap D(q, r_2)$. Note that the boundary case happens when $r_1 = \sqrt{2}r_2$ and there exists *exactly* one such segment.

Case 1: $r_1 \leq \sqrt{2}r_2$. We denote the intersection points between the boundaries of $D(o, r_1)$ and $D(q, r_2)$ by a and b, as shown in Figure 2(a). Let c be the intersection point between segment $\overline{qo}$ and segment $\overline{ab}$. Let $x = d(a, c) = d(b, c)$ and $y = d(c, q)$. Since $\triangle_{ocb}$ and $\triangle_{qcb}$ are right-angled triangles, we know $x^2 + (r_2 - y)^2 = r_1^2$ and $y^2 + x^2 = r_2^2$ by the *hypothesis theorem*. By solving these two equations, we obtain $x = \sqrt{r_1^2 - r_1^4/4r_2^2}$ and thus $d(a, b) = 2x = 2\sqrt{r_1^2 - r_1^4/4r_2^2}$. In this case, it can be verified that $\max_{o_1,o_2 \in S'} d(o_1, o_2) \leq d(a, b)$ (since all objects in S' are in $D(o, r_1) \cap D(q, r_2)$, as shown in the shaded area of Figure 2(a)) and hence $cost(S') \leq r_2 + 2\sqrt{r_1^2 - r_1^4/4r_2^2}$. Therefore,

$$\frac{cost(S')}{cost(S_o)} \leq \frac{r_2 + 2\sqrt{r_1^2 - r_1^4/4r_2^2}}{r_2 + r_1} = 1 + \frac{2\sqrt{1 - r_1^2/4r_2^2} - 1}{r_2/r_1 + 1}$$

Let $z = r_1/r_2$. Thus, $\frac{cost(S')}{cost(S_o)} \leq 1 + \frac{2\sqrt{1-z^2/4}-1}{1/z+1}$. Since $r_1 \leq \sqrt{2}r_2$, we have $z \in (0, \sqrt{2}]$ [1]. We define $f(z) = 1 + \frac{2\sqrt{1-z^2/4}-1}{1/z+1}$ on $\{z | z \in (0, \sqrt{2}]\}$. It could be verified that $f(z)$ is monotonically increasing on $(0, 0.875)$ and is monotonically decreasing on $(0.875, \sqrt{2}]$. Thus, $f(z) \leq f(0.875) < 1.375$. Therefore,

$$\frac{cost(S')}{cost(S_o)} \leq f(z) \leq 1.375$$

Case 2: $r_1 > \sqrt{2}r_2$. Let $\overline{ab}$ be any segment linking two points at the boundary of $D(q, r_2)$ which has its length equal to $2r_2$ and falls in $D(o, r_1) \cap D(q, r_2)$. For illustration, consider Figure 2(b). That is, $d(a, b) = 2r_2$. Similar to Case 1, it could be verified that $\max_{o_1,o_2 \in S'} d(o_1, o_2) \leq d(a, b) = 2r_2$. Thus, $cost(S') \leq r_2 + 2r_2$. Therefore,

$$\frac{cost(S')}{cost(S_o)} \leq \frac{r_2 + 2r_2}{r_2 + r_1} = \frac{1 + 2}{1 + r_1/r_2} \leq \frac{1 + 2}{1 + \sqrt{2}} < 1.25$$

Thus, by combining Case 1 and Case 2, we have $cost(S') \leq 1.375 \cdot cost(S_o)$, which completes the proof. □

Implementation and Time Complexity. We also adopt the IR-tree built on $\mathcal{O}$ to support the NN query and the range query.

Let n_1 be the number of iterations in MaxSum-Appro (lines 2-10 in Algorithm 3) and γ be the cost of executing an iteration. Then, the time complexity of MaxSum-Appro is $O(n_1 \cdot \gamma)$. Note that γ is dominated by the step of finding the o-neighborhood feasible set (line 6) whose cost is bounded by $O(|q.\psi| \cdot \log |\mathcal{O}|)$ (it issues at most $|q.\psi - o.\psi|$ NN queries each of which takes $O(\log |\mathcal{O}|)$ time). Thus, $\gamma = O(|q.\psi| \cdot \log |\mathcal{O}|)$. Therefore, the time complexity of MaxSum-Appro is $O(n_1 \cdot |q.\psi| \cdot \log |\mathcal{O}|)$ where $n_1 << |\mathcal{O}_q|$. Note that the worst-case time complexity of MaxSum-Appro is $O(|\mathcal{O}_q| \cdot |q.\psi| \cdot \log |\mathcal{O}|)$, which is the same as that of Cao-Appro2.

4. ALGORITHMS FOR DIA-COSKQ

In this section, we propose two algorithms, Dia-Exact and Dia-Appro, for Dia-CoSKQ. Similarly, in this section, for clarity, we simply write $cost_{Dia}(\cdot)$ as $cost(\cdot)$ if the context of the cost function is clear.

4.1 Finding Optimal Solution

Interestingly, we can adopt the same MaxSum-Exact algorithm (Algorithm 1) by replacing the cost measurement from the maximum sum cost to the diameter cost. We call this algorithm *Dia-Exact*. The reason is that we can still use the query distance owner and the pairwise distance owners of a set S' to be found to find the optimal solution for Dia-CoSKQ. Next, we explain the reason in detail.

Consider the diameter cost. Given a set S' of objects in $\mathcal{O}$, we have $cost(S') = \max_{o',o'' \in S' \cup \{o_q\}} d(o', o'')$. Clearly, the (diameter) cost of a set S' can be dominated (or determined) by two pairwise distance owners of $S' \cup \{o_q\}$ (not S' used in the maximum sum cost), which form a distance owner group (for Dia-CoSKQ). It is similar to the maximum sum cost of a set S' which is dominated by the query distance owner of S' and two pairwise distance owners of S'. But, there are two differences. The first difference is that the diameter cost is dominated by the pairwise distance owners only (without the query distance owner). The second difference is that the pairwise distance owners used for the diameter cost are based on the set $S' \cup \{o_q\}$ instead of S'.

Based on the above observations, we *directly* adapt the distance owner-driven approach as follows. This approach maintains a variable S storing the best feasible set found so far. Initially, S is set to a feasible set. This involves three major steps.

- *Step 1 (Pairwise Distance Owner Finding):* Select two objects, o' and o'', in $\mathcal{O}_q \cup \{o_q\}$ to take the roles of the pairwise distance owners of the set $S' \cup \{o_q\}$ where S' is to be found. Note that o' and o'' form a distance owner group.

[1] The interval $(0, \sqrt{2}]$ does not include the boundary case where $z = 0$ (i.e., $r_1 = 0$). In this case, we have $cost(S')/cost(S_o) = 1$.

- *Step 2 (Sub-Optimal Feasible Set Finding):* Find a set S' of objects in $\mathcal{O}_q$ such that the pairwise distance owners of $S' \cup \{o_q\}$ are o' and o'' (if any), and update S with S' if $cost(S') < cost(S)$.
- *Step 3 (Iterative Step):* Repeat Step 1 which finds another distance group, and continue with Step 2 until all distance owner groups have been traversed.

Interestingly, Step 1 which *originally* finds two objects to take the roles of the two pairwise distance owners *based on* $S' \cup \{o_q\}$ can be *refined* to a number of sub-steps of finding two objects to take the roles of the two pairwise distance owners *based on* S' simply (not $S' \cup \{o_q\}$) and finding an object to take the role of the query distance owner *based on* S'. This refinement can be explained by the following observation.

OBSERVATION 1. *Let S' be the feasible set. The pairwise distance owners of $S' \cup \{o_q\}$ are either (1) o_q and the query distance owner of S' or (2) the pairwise distance owners of S'.* □

Suppose that o takes the role of the query distance owner of S' to be found, and o_1 and o_2 take the roles of the two pairwise distance owners of S'.

Observation 1 involves two cases. In Case (1) of Observation 1, we know that the pairwise distance owners of $S' \cup \{o_q\}$ are o_q and the query distance owner o of S'. In this case, we deduce that $d(o_1, o_2) \leq d(o, q) (= d(o, o_q))$.

In Case (2) of Observation 1, we know that the pairwise distance owners of $S' \cup \{o_q\}$ are the pairwise distance owners of S', say o_1 and o_2. In this case, $d(o, q) \leq d(o_1, o_2)$.

In conclusion, if we know that $d(o_1, o_2) \leq d(o, q)$, then o_q and o are the pairwise distance owners of $S' \cup \{o_q\}$. Otherwise, o_1 and o_2 are the pairwise distance owners of $S' \cup \{o_q\}$.

Thus, Step 1 can be refined with the following three sub-steps.

- *Step 1(a) (Query Distance Owner Finding):* Select an object o in $\mathcal{O}_q$ to take the role of the query distance owner of a set S' to be found.
- *Step 1(b) (Pairwise Distance Owner Finding):* Select two objects o_1 and o_2 in $D(q, d(o, q))$ to take the roles of the pairwise distance owners *of the set* S' to be found.
- *Step 1(c) (Pairwise Distance Owner Determination):* If $d(o, q) \geq d(o_1, o_2)$, assign to o and o_q the roles of pairwise distance owners of $S' \cup \{o_q\}$; otherwise, assign the roles to o_1 and o_2.

With this refinement, the distance owner-driven approach still has its similar pruning features under the diameter cost. Specifically, Property 1 and Property 2 used for MaxSum-CoSKQ have their counterparts used for Dia-CoSKQ as Property 3 and Property 4, respectively.

PROPERTY 3 (PRUNING). *Let S' be a feasible set. If o is the query distance owner of S', then the two pairwise distance owners of $S' \cup \{o_q\}$ are inside $D(q, d(o, q))$.* □

PROPERTY 4 (PRUNING). *Let S' be a feasible set, o be the query distance owner of S', and o_1 and o_2 be the two pairwise distance owners of $S' \cup \{o_q\}$. Then all objects in S' fall in $D(q, d(o, q)) \cap D(o_1, d(o_1, o_2)) \cap D(o_2, d(o_1, o_2))$.* □

Similar to the maximum sum cost, when the diameter cost is used, the object to be found in Step 1(a) is fetched based on the proximity to the query point q. The proximity is also related to the closest possible query distance owner (Lemma 2) and the farthest possible query distance owner (Lemma 3). It is easy to verify that Lemma 2 and Lemma 3 still hold when the cost measurement is changed from the maximum sum cost to the diameter cost. Thus, Lemma 4, which states that the *ring* is the region containing the query distance owners to be considered, still holds.

Algorithm 4 Algorithm *Dia-Exact*

Input: query q and a set $\mathcal{O}$ of objects

```
1: S ← N(q)
2: while there is an "un-processed" relevant object o in R(S) do
3:    // Step 1(a) (Query Distance Owner Finding)
4:    o ← the nearest "un-processed" relevant object in R(S)
5:    // Step 1(b) (Pairwise Distance Owner Finding)
6:    D ← the q-disk with its radius equal to d(o, q)
7:    P ← a set of all pairs (o1, o2) where o1 and o2 are in D
8:    for each (o1, o2) ∈ P in ascending order of d(o1, o2) do
9:       // Step 1(c) (Pairwise Distance Owner Determination)
10:      if d(o, q) ≥ d(o1, o2) then o' ← o; o'' ← o_q
11:      else o' ← o1; o'' ← o2
12:      // Step 2 (Sub-Optimal Feasible Set Finding)
13:      if there exists a feasible set S' in D which is (o, o', o'')-
         owner consistent then
14:         if cost(S') < cost(S) then
15:            S ← S'; break
16:   // Step 3 (Iterative Process)
17:   mark o as "processed"
18: return S
```

In summary, we present the algorithm for finding the optimal solution of Dia-CoSKQ in Algorithm 4 (which is quite similar to Algorithm 1) except that we need to determine the pairwise distance owner of $S' \cup \{o_q\}$ (in Step 1(c)) which cannot be found in MaxSum-CoSKQ.

THEOREM 3. *Dia-Exact returns a feasible set with the smallest cost for the Dia-CoSKQ problem.* □

PROOF. Let S_o be one of the feasible set with the smallest cost. Let o be the query distance owner of S_o, and let o_1 and o_2 be the two pairwise distance owners of S_o. First, o is inside $R(S)$ (Lemma 4). Thus, there exists an iteration where o is processed. When o is processed, pair (o_1, o_2) must be included in P (Property 3). There are two cases. Case 1: $d(o, q) \geq d(o_1, o_2)$. In this case, any feasible set S' that is (o, o, o_q)-owner consistent is retrieved and used to update S (there must exist some since S_o is (o, o, o_q)-owner consistent). Thus, the resulting S has its cost equal to $d(o, q) = cost(S_o)$. Case 2: $d(o, q) < d(o_1, o_2)$. In this case, any feasible set S' that is (o, o_1, o_2)-owner consistent is retrieved and used to update S (there must exist some since S_o is (o, o_1, o_2)-owner consistent). Thus, the resulting S has its cost equal to $d(o_1, o_2) = cost(S_o)$. In either case, S will not be updated anymore since it has the smallest cost (i.e., $cost(S_o)$) and thus it is the final output. □

Same as Section 3.1.4, in Dia-Exact, we have the self-iteration computation strategy and the cross-iteration computation strategy.

Self-Iteration Computation Strategy: Consider an iteration where the query distance owner for this iteration is o. We can use the same mechanism described in Section 3.1.4 after d_{min} and d_{max} are updated from $d(o, q) - \min\{d(o_1, q), d(o_2, q)\}$ and $cost(S) - d(o, q)$ to $d(o, q)$ and $cost(S)$, respectively. All pruning properties still hold.

Note that d_{min} (which is originally set to $d(o, q) - \min\{d(o_1, q), d(o_2, q)\}$ in MaxSum-CoSKQ) is based on the triangle inequality (Lemma 5), which means that it can be used for

pruning in both MaxSum-CoSKQ and Dia-CoSKQ. However, in Dia-CoSKQ, d_{min} can be updated to a tighter value as $d(o,q)$ since all pairs with their pairwise distances smaller than $d(o,q)$ cannot take the roles of the pairwise distance owners of $S' \cup \{o_q\}$.

Cross-Iteration Computation Strategy: We use the same information reuse techniques as in Section 3.1.4 for Dia-Exact since the updated d_{min} (i.e., $d(o,q)$) is monotonically increasing and the updated d_{max} (i.e., $cost(S)$) is monotonically decreasing with more iterations. Thus, the pairs pruned in P at the previous iterations need not be considered in the later iterations.

Time Complexity. It could be verified that the time complexity of Dia-Exact is the same as that of MaxSum-Exact.

4.2 Finding Approximate Solution

In this section, we propose a $\sqrt{3}$-factor approximate algorithm which is exactly the same as Algorithm 3 but the cost measurement used is the diameter cost. This algorithm is called *Dia-Appro*.

Theoretical Analysis. Although the set S returned by *Dia-Appro* may have a larger cost than the optimal set S_o, it has an approximate factor of $\sqrt{3}$.

THEOREM 4. *Dia-Appro gives a $\sqrt{3}$-factor approximation for the Dia-CoSKQ problem.* □

PROOF. We use the same notations as defined in the proof of Theorem 2.

Consider $cost(S_o)$. Similar to the proof of Theorem 2, we have $\max_{o'_1,o'_2 \in S_o} d(o'_1,o'_2) \geq d(o,o_f) = r_1$. Recall that $\max_{o' \in S_o} d(o',q) = d(o,q) = r_2$. As a result, we have $cost(S_o) = \max\{\max_{o' \in S_o} d(o',q), \max_{o'_1,o'_2 \in S_o} d(o'_1,o'_2)\} \geq \max\{r_2, r_1\}$.

According to the Dia-Appro algorithm, we have $cost(S) \leq cost(S')$. The remaining part of the proof shows that $cost(S') \leq \sqrt{3} \cdot cost(S_o)$ which further implies $cost(S) \leq \sqrt{3} \cdot cost(S_o)$.

Same as the proof of Theorem 2, we consider two cases of r_1.

Case 1: $r_1 \leq \sqrt{2} r_2$. This case corresponds to Figure 2(a). It can be verified that $\max_{o'_1,o'_2 \in S'} d(o'_1,o'_2) \leq d(a,b) = 2\sqrt{r_1^2 - r_1^4/4r_2^2}$ (since all objects in S' fall in $D(o,r_1) \cap D(q,r_2)$ as shown by the shaded area). Recall $\max_{o' \in S'} d(o,q) = r_2$. As a result, we have $cost(S') \leq \max\{r_2, 2\sqrt{r_1^2 - r_1^4/4r_2^2}\}$.

We further consider three sub-cases under Case 1 based on the relationship among r_1, r_2 and $2\sqrt{r_1^2 - r_1^4/4r_2^2}$.

Case 1(a): $r_1 \leq \sqrt{2-\sqrt{3}} r_2$. In this case, we have $r_2 > r_1$ and $r_2 \geq 2\sqrt{r_1^2 - r_1^4/4r_2^2}$. Thus, $cost(S_o) \geq \max\{r_2, r_1\} = r_2$ and $cost(S') \leq \max\{r_2, 2\sqrt{r_1^2 - r_1^4/4r_2^2}\} = r_2$ Therefore,

$$\frac{cost(S')}{cost(S_o)} \leq \frac{r_2}{r_2} = 1$$

Case 1(b): $\sqrt{2-\sqrt{3}} r_2 \leq r_1 \leq r_2$. In this case, we have $r_2 \geq r_1$ and $2\sqrt{r_1^2 - r_1^4/4r_2^2} > r_2$. Thus, $cost(S_o) \geq r_2$ and $cost(S') \leq 2\sqrt{r_1^2 - r_1^4/4r_2^2}$. Therefore,

$$\frac{cost(S')}{cost(S_o)} \leq \frac{2\sqrt{r_1^2 - r_1^4/4r_2^2}}{r_2} = \sqrt{4(\frac{r_1}{r_2})^2 - (\frac{r_1}{r_2})^4} \quad (4)$$

Note that function $f(z) = \sqrt{4z^2 - z^4}$ is monotonically increasing on $(\sqrt{2-\sqrt{3}}, 1]$. Since $\frac{r_1}{r_2} \in (\sqrt{2-\sqrt{3}}, 1]$, Thus, we have

$$\frac{cost(S')}{cost(S_o)} \leq \sqrt{4(1)^2 - (1)^4} = \sqrt{3}$$

Case 1(c): $r_2 < r_1 \leq \sqrt{2} r_2$. In this case, we have $r_2 < r_1$ and $2\sqrt{r_1^2 - r_1^4/4r_2^2} > r_2$. Thus, $cost(S_o) \geq r_1$ and $cost(S') \leq 2\sqrt{r_1^2 - r_1^4/4r_2^2}$. Therefore,

$$\frac{cost(S')}{cost(S_o)} \leq \frac{2\sqrt{r_1^2 - r_1^4/4r_2^2}}{r_1} = \sqrt{4 - (r_1/r_2)^2} < \sqrt{3}$$

Case 2: $r_1 > \sqrt{2} r_2$. This case corresponds to Figure 2 (b). In this case, $d(a,b) = 2r_2$. Similar to Case 1, we have $\max_{o_1,o_2 \in S'} d(o_1,o_2) \leq d(a,b) = 2r_2$. Therefore,

$$\frac{cost(S')}{cost(S_o)} \leq \frac{\max\{r_2, 2r_2\}}{\max\{r_1, r_2\}} = \frac{2r_2}{r_1} < \frac{2r_2}{\sqrt{2}r_2} = \sqrt{2}$$

In view of above discussion, we know that $cost(S')/cost(S_o) \leq \sqrt{3}$, which completes the proof. □

Time Complexity. Since Dia-Appro is identical to MaxSum-Appro except that Dia-Appro adopts a different cost measurement, Dia-Appro has the same complexity as MaxSum-Appro.

4.3 Adaptions of Existing Solutions

In this section, we adapt the existing solutions in [4], which are originally designed for MaxSum-CoSKQ, for Dia-CoSKQ.

Cao-Exact. Cao-Exact is a best-first search method based on the object space and thus its applicability is independent of the cost measurement used in the CoSKQ problem. Therefore, Cao-Exact can be directly applied to Dia-CoSKQ by replacing the cost measurement with the diameter cost. However, due to its prohibitively huge search space, Cao-Exact is not scalable to large datasets.

Cao-Appro1 & Cao-Appro2. We can directly adopt Cao-Appro1 and Cao-Appro2 for Dia-CoSKQ by replacing the maximum sum cost with the diameter cost.

According to [4], the approximation factors of Cao-Appro1 and Cao-Appro2 are 3 and 2, respectively, for MaxSum-CoSKQ. In the following, we prove that both Cao-Appro1 and Cao-Appro2 give 2-factor approximations for Dia-CoSKQ.

LEMMA 7. *Cao-Appro1 and Cao-Appro2 give 2-factor approximations for Dia-CoSKQ.* □

PROOF. First, we prove that the approximation ratio of Cao-Appro1 is 2.

Let S be the set returned by Cao-Appro1 and S_o be the optimal set. Let o_f be the object in S that is the farthest from q, i.e., $d(o_f,q) = \max_{o \in S} d(o,q)$. First, we have $cost(S_o) \geq d(o_f,q)$. Second, for any two objects o_1 and o_2 in S, we have $d(o_1,o_2) \leq d(o_1,q) + d(o_2,q) \leq 2 \cdot d(o_f,q)$ by the *triangle inequality*. Therefore, $cost(S) \leq \max\{d(o_f,q), 2 \cdot d(o_f,q)\} = 2 \cdot d(o_f,q)$. As a result, we know $cost(S)/cost(S_o) \leq 2$.

Since the solution returned by Cao-Appro2 is no worse than that returned by Cao-Appro1, the approximation ratio of Cao-Appro2 is also bounded by 2.

Furthermore, we show that Cao-Appro2 cannot provide better error guarantees by constructing a problem instance where the approximation ratio of Cao-Appro2 is infinitely close to 2. Due to page limit, we refer the reader to [16] for the problem instance. □

Thus, among all known approximate algorithms for Dia-CoSKQ, our Dia-Appro provides the best constant-factor approximation.

5. EMPIRICAL STUDIES

5.1 Experimental Set-up

Datasets. We used the real datasets adopted in [4], namely Hotel, Web and GN. Dataset Hotel corresponds to a set of hotels in

Statistics	GN	Web	Hotel
Number of objects	1,868,821	579,727	20,790
Number of unique words	222,409	2,899,175	602
Number of words	18,374,228	249,132,883	80,845

Table 1: Real datasets

the U.S. (www.allstays.com), each of which is associated with its location and a set of words that describe the hotel (e.g., restaurant and pool). Dataset Web was created from two real datasets. The first one, named WEBSPAMUK2007[2], corresponds to a set of web documents. The second one is a set of spatial objects, named TigerCensusBlock[3], which corresponds to a set of census blocks in Iowa, Kansas, Missouri and Nebraska. Specifically, Web consists of the spatial objects in TigerCensusBlock, each of which is associated with a document randomly selected from WEBSPAMUK2007. Dataset GN was collected from the U.S. Board on Geographic Names (geonames.usgs.gov). Each object in GN is a 2D location which is associated with a set of keywords describing it (e.g., a geographic name like valley).

Query Generation. Given a dataset $\mathcal{O}$ and a positive integer k, we generated a query q with the size of its keyword set equal to k as in [4]. For the $q.\lambda$ part, we randomly picked a location from the data space of $\mathcal{O}$. For the $q.\psi$ part, we first sorted all the keywords that are associated with the objects in $\mathcal{O}$ in descending order of their frequencies and then randomly picked k keywords among all keywords each of which has its *percentile rank* within range $[10, 40]$ by default. Note that in this way, each of the keywords in $q.\psi$ has a relatively high frequency.

Algorithms. For MaxSum-CoSKQ, we consider 2 exact algorithms, namely MaxSum-Exact and Cao-Exact, and 3 approximate algorithms, namely MaxSum-Appro, Cao-Appro1 and Cao-Appro2. For Dia-CoSKQ, we consider 2 exact algorithms, namely Dia-Exact and Cao-Exact (the adaption), and 3 approximate algorithms, namely Dia-Appro, Cao-Appro1 and Cao-Appro2. All algorithms were implemented in C/C++.

Our experiments were conducted on a Linux platform with a 2.66GHz machine and 4GB RAM.

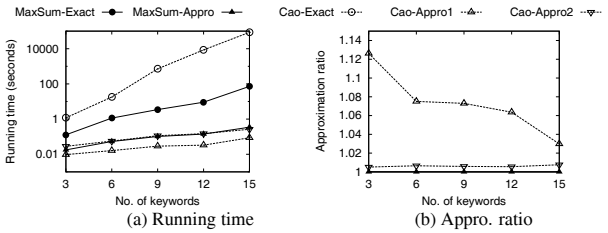
5.2 Experimental Results

We consider 2 measurements, the running time and the approximation ratio (for approximate algorithms only). For each set of settings, we generated 50 queries, ran the algorithms with each of these 50 queries, and averaged the experimental measurements.

5.2.1 Experiments for MaxSum-CoSKQ

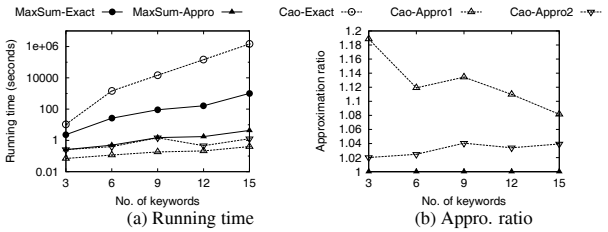
Effect of $|q.\psi|$. We generated 5 types of queries with different values of $|q.\psi|$. The values we used are 3, 6, 9, 12 and 15. The results on the dataset GN are shown in Figure 3. According to Figure 3(a), our MaxSum-Exact is faster than Cao-Exact by 1-3 orders of magnitude. When $|q.\psi|$ increases, the running time gap between MaxSum-Exact and Cao-Exact increases. Besides, MaxSum-Appro and Cao-Appro2 have comparable running time, which verified our theoretical analysis that MaxSum-Appro and Cao-Appro2 have the same worst-case time complexity. Cao-Appro1 runs the fastest due to its simplicity. According to Figure 3(b), the approximation ratio of our MaxSum-Appro algorithm is near to 1, which shows that the accuracy of MaxSum-Appro is extremely high in practical. We note here that the approximation ratio in the figure corresponds to the average over 50 queries, among which, the approximation ratio of MaxSum-Appro is exactly 1 for most queries (e.g., more than 45). As a result, the approximation ratio of MaxSum-Appro in the figures is always near to 1. Consistent to our theoretical results, the approximation ratios of Cao-Appro1 and Cao-Appro2 are larger than that of MaxSum-Appro.

We have similar results on Web (Figure 4) and Hotel (Figure 5).



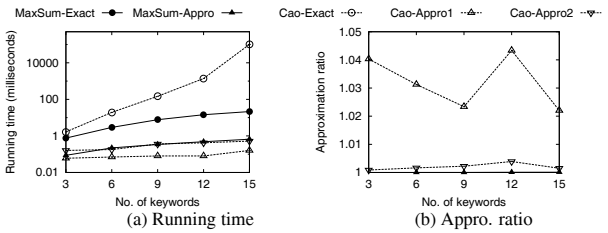
(a) Running time (b) Appro. ratio

Figure 3: Effect of $|q.\psi|$ (GN, MaxSum-CoSKQ)



(a) Running time (b) Appro. ratio

Figure 4: Effect of $|q.\psi|$ (Web, MaxSum-CoSKQ)



(a) Running time (b) Appro. ratio

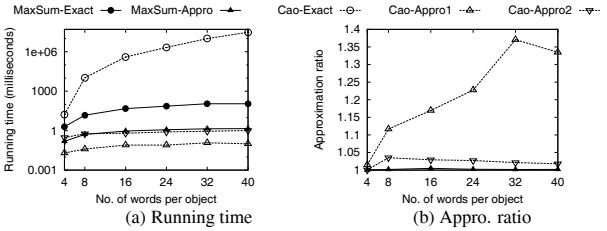
Figure 5: Effect of $|q.\psi|$ (Hotel, MaxSum-CoSKQ)

Effect of average $|o.\psi|$. Our experiments were based on dataset Hotel whose average size of a keyword set of an object ($|o.\psi|$) is nearly 4 (i.e., 80,845/20,790). We generated a set of several datasets based on dataset Hotel such that the average sizes (i.e., average $|o.\psi|$'s) are equal to $4 \cdot i$ for some integers i. To generate a dataset with its average $|o.\psi|$ equal to $4 \cdot i$, we proceed with $i - 1$ rounds. At each round, for each object o in dataset Hotel, we randomly pick another object o' and update $o.\psi$ to be $o.\psi \cup o'.\psi$. It could be verified that the average $|o.\psi|$ of the resulting dataset is nearly $4 \cdot i$. In our experiments, we vary i by choosing one of the values in $\{1, 2, 4, 6, 8, 10\}$. Note that $i = 1$ means that the resulting dataset is exactly dataset Hotel.

The results are shown in Figure 6. According to Figure 6(a), the running times of all algorithms increase when the average $|o.\psi|$ increases. The reason is that when the average $|o.\psi|$ increases, the number of relevant objects ($|\mathcal{O}_q|$) in the dataset would probably increase, which further affects the running times of the algorihtms. Since all algorithms except for Cao-Appro1 have their time complexities involving $|\mathcal{O}_q|$. Cao-Appro1, though has its time complexity independent of $|\mathcal{O}_q|$, has its NN queries affected by $|\mathcal{O}_q|$: the larger $|\mathcal{O}_q|$ is, the more expensive the NN query would probably be. Besides, it is worth mentioning that when the average $|o.\psi|$ increases, the increase rate of the running time of Cao-Exact is significantly larger than those of the other algorithms including MaxSum-Exact. This is because Cao-Exact is based on the search

[2]http://barcelona.research.yahoo.net/webspam/datasets/uk2007
[3]http://www.rtreeportal.org

space of the set of all possible feasible sets whose size increases rapidly with $|\mathcal{O}_q|$ ($|\mathcal{O}_q|^{|q.\psi|}$). Thus, Cao-Exact is not scalable on datasets with a large average $|o.\psi|$. According to Figure 6(b), the average $|o.\psi|$ has no obvious trend on the approximation ratios of the approximate algorithms. Besides, MaxSum-Appro with its approximation ratio near to 1 always keeps its accuracy superiority over other approximate algorithms.

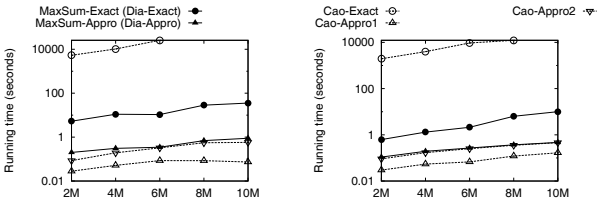


(a) Running time (b) Appro. ratio

Figure 6: Effect of average $|o.\psi|$ (MaxSum-CoSKQ)

Scalability Test. We conducted a scalability test on the algorithms with 5 synthetic datasets with their sizes varying from 2M to 10M. The synthetic datasets were generated from a smaller dataset GN. To generate a dataset $\mathcal{O}$ with its size equal to n, we first inserted all the objects from dataset GN into $\mathcal{O}$ and then repeatedly created objects in $\mathcal{O}$ such that $\mathcal{O}$ has a similar spatial distribution as dataset GN until $|\mathcal{O}| = n$. For each newly created object o in $\mathcal{O}$, we randomly pick a document from WEBSPAMUK2007 and use it as $o.\psi$.

The results are shown in Figure 7(a), where we do not show the running time of the algorithm if it runs more than 10 days or out of memory. According to these results, both our exact algorithm (MaxSum-Exact) and our approximate algorithm (MaxSum-Appro) are scalable to large datasets with millions of objects. For example, in a dataset with size equal to 10M, MaxSum-Exact ran less than 100s and MaxSum-Appro ran in real-time. In contrast, Cao-Exact is not scalable. In particular, in our experiments, Cao-Exact took more than 1 day on a dataset with size equal to 6M and it took more than 10 days on a dataset with size equal to 8M.



(a) MaxSum-CoSKQ (b) Dia-CoSKQ

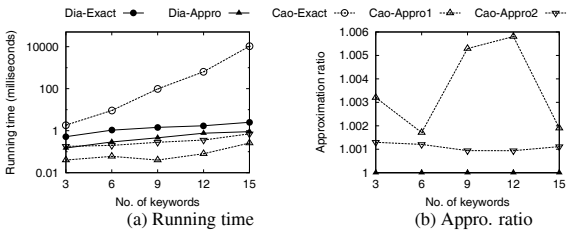
Figure 7: Scalability Test

5.2.2 *Experiments for Dia-CoSKQ*

Effect of $|q.\psi|$. The results on dataset GN is shown in Figure 8. According to Figure 8(a), our Dia-Exact is faster than Cao-Exact by 1-4 orders of magnitude. When $|q.\psi|$ increases, the running time gap between Dia-Exact and Cao-Exact increases. Besides, Dia-Appro and Cao-Appro2 have comparable running times. According to Figure 8(b), similar to MaxSum-Appro (for MaxSum-CoSkQ), the approximation ratio of Dia-Appro is near to 1 (for Dia-CoSKQ), which is better than those of Cao-Appro1 and Cao-Appro2.

The results on datasets Web and Hotel are similar and thus they are omitted here due to the page limit.

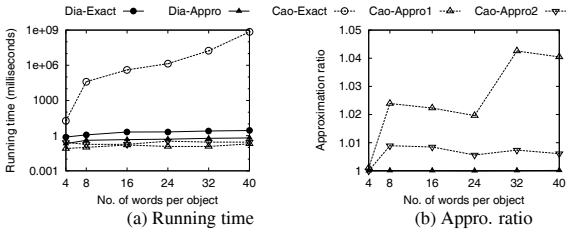
Effect of Average $|o.\psi|$. Similar to the experiments for MaxSum-CoSKQ, we generated a set of datasets by varying their average



(a) Running time (b) Appro. ratio

Figure 8: Effect of $|q.\psi|$ (GN, Dia-CoSKQ)

$|o.\psi|$ values. The results are shown in Figure 9. According to Figure 9(a), when the average $|o.\psi|$ increases, the running time of Cao-Exact increases significantly while the running times of other algorithms are only slightly affected. This is similar to the case for MaxSum-CoSKQ and the explanation for MaxSum-CoSKQ as we discussed previously could be applied here for Dia-CoSKQ. According to Figure 9(b), the average $|o.\psi|$ value has no obvious trend on the accuracy of the approximate algorithms.



(a) Running time (b) Appro. ratio

Figure 9: Effect of average $|o.\psi|$ (Dia-CoSKQ)

Scalability Test. We conducted a scalability test on the algorithms for Dia-CosKQ with the same synthetic datasets used in the scalability test for MaxSum-CoSKQ.

The results are shown in Figure 7(b), where we do not show the running time of the algorithm if it runs more than 10 days or out of memory. According to these results, both our exact algorithm (Dia-Exact) and our approximate algorithm (Dia-Appro) are scalable to large datasets with millions of objects. In contrast, Cao-Exact is not scalable to large datasets.

Conclusion: MaxSum-Exact (Dia-Exact) runs faster than Cao-Exact by several orders of magnitude for MaxSum-CoSKQ (Dia-CoSKQ). Besides, MaxSum-Exact (Dia-Exact) is scalable in terms of $|\mathcal{O}|$ as well as the average $|o.\psi|$ but Cao-Exact is not. Our MaxSum-Appro (Dia-Appro) has a better accuracy while having comparable running time as those existing approximate algorithms.

6. RELATED WORK

Many types of spatial keyword query have been proposed in the literature. Most of them are different from CoSKQ studied in this paper since they use a single object to cover all keywords specified in the query but CoSKQ uses multiple objects collectively for the same purpose. We review these spatial keyword queries as follows.

A *spatial keyword top-k query* [8] finds top-k objects where the ranking function takes both the spatial proximity and the textual relevance of the objects into consideration. This branch includes [8, 19, 14] (Euclidean space), [20] (road networks), [21, 9] (trajectory databases), and [24] (moving objects). A common technique shared by these studies is to design a hybrid indexing structure, which captures both the spatial proximity and the textual information of the objects. The IR-tree adopted by us for NN queries and range queries was proposed in [8].

A *spatial keyword k-NN query* [11] finds the k-NNs from the query location, each of which contains the set of keywords speci-

fied in the query. That is, unlike the keywords in the spatial keyword top-k queries, which are used as a *soft* constraint, the keywords in the spatial keyword k-NN queries are used as a *hard* constraint. This branch includes [11, 5, 23].

A *spatial keyword range query* [22, 28, 7] takes a region and a set of keywords as input and finds the objects each of which falls in the region and contains the set of keywords. Same as the spatial keyword k-NN queries, the keywords are used as a hard constraint. Usually, they combine a spatial index (e.g., R-tree and Space Filling Curve (SFC)) and a textual index (e.g., inverted file) for query processing.

A *spatial keyword reverse top-k query* [17] finds the set of objects whose spatial keyword top-k query results include the query. Note that in this case, an object which consists of a location and a set of keywords could be regarded as a query which also consists of a location and a set of keywords and vice versa.

An *mCK query* [25, 26] is a spatial keyword query that is very similar to CoSKQ. An mCK query takes m keywords as input and finds m objects with the smallest *diameter* that cover the m keywords specified in the query. Though both the mCK query and CoSKQ use a set of objects for covering a set of keywords collectively, they are different. In the context of an mCK query, it is assumed that each object is associated with a single keyword while in the context of CoSKQ, each object is associated with a set of multiple keywords. Besides, an mCK query only takes a set of keywords as input while our CoSKQ query takes not only a set of keywords but also a query location as an input.

CoSKQ was first studied in [4]. Under the maximum sum cost function, as we described, [4] proposed an exact algorithm and two approximate algorithms. However, the exact algorithm is not scalable to large datasets and the two approximate algorithms cannot guarantee near-to-optimal solutions. In this paper, we propose an efficient exact algorithm and an approximate algorithm with better approximate factor for MaxSum-CoSKQ. Besides, in this paper, we also propose another cost function called the diameter function which is new and has not been studied in [4].

7. CONCLUSION

In this paper, we studied two types of the CoSKQ problem, namely MaxSum-CoSKQ and Dia-CoSKQ. MaxSum-CoSKQ is a CoSKQ problem using the existing maximum sum cost, which is NP-hard. We designed two algorithms for MaxSum-CoSKQ, MaxSum-Exact and MaxSum-Appro. MaxSum-Exact is an exact algorithm which significantly outperforms its existing competitor in terms of both efficiency and scalability and MaxSum-Appro is an approximate algorithm which improves the best-known constant approximation factor from 2 to 1.375. We also proposed a new cost function and the CoSKQ problem using this function is Dia-CoSKQ. We designed two algorithms for Dia-CoSKQ, Dia-Exact and Dia-Appro. Dia-Exact is an exact algorithm while Dia-Appro is a $\sqrt{3}$-factor approximate algorithm. Extensive experiments were conducted which verified our theoretical findings and algorithms.

There are several interesting future research directions. One direction is to find the feasible set with the smallest *cost per object*. Another direction is to define the cost function based on the shortest route that traverse all objects in the set. It is also interesting to to study CoSKQ when the query point is moving.

Acknowledgements: We appreciate the help from Cao et al. for passing us their real datasets [4]. We are grateful to the anonymous reviewers for their valuable comments on this paper. The research of Cheng Long and Raymond Chi-Wing Wong is supported by grants DAG12EG077 and FSGRF13EG27.

8. REFERENCES

[1] A. Aggarwal, H. Imai, N. Katoh, and S. Suri. Finding k points with minimum diameter and related problems. *Journal of Algorithms*, 12(1):38–56, 1991.

[2] E. M. Arkiny and R. Hassinz. Minimum diameter covering problems. *Networks*, 36(3), 2000.

[3] X. Cao, G. Cong, and C. S. Jensen. Retrieving top-k prestige-based relevant spatial web objects. *VLDB*, 3(1-2):373–384, 2010.

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

[5] A. Cary, O. Wolfson, and N. Rishe. Efficient and scalable method for processing top-k spatial boolean queries. In *Scientific and Statistical Database Management*, pages 87–95. Springer, 2010.

[6] T. M. Chan. A dynamic data structure for 3-d convex hulls and 2-d nearest neighbor queries. In *SODA*, 2006.

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

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

[9] 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.

[10] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. Computational geometry: Algorithms and applications. In *Springer*, 2000.

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

[12] G. R. Hjaltason and H. Samet. Distance browsing in spatial databases. *ACM TODS*, 24(2):265–318, 1999.

[13] T. Lappas, K. Liu, and E. Terzi. Finding a team of experts in social networks. In *SIGKDD*, pages 467–476. ACM, 2009.

[14] Z. Li, K. Lee, B. Zheng, W. Lee, D. Lee, and X. Wang. Ir-tree: An efficient index for geographic document search. *TKDE*, 2011.

[15] W. Liu, W. Sun, C. Chen, Y. Huang, Y. Jing, and K. Chen. Circle of friend query in geo-social networks. In *Database Systems for Advanced Applications*, pages 126–137. Springer, 2012.

[16] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu. Collective spatial keyword queries:a distance owner-driven approach (technical report). In *http://www.cse.ust.hk/~raywong/paper/coskq-technical.pdf*, 2013.

[17] J. Lu, Y. Lu, and G. Cong. Reverse spatial and textual k nearest neighbor search. In *SIGMOD*, pages 349–360. ACM, 2011.

[18] K. Mulmuley. Computational geometry: An introduction through randomized algorithms. In *Prentice Hall*, 1993.

[19] J. Rocha, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. *Advances in Spatial and Temporal Databases*, pages 205–222, 2011.

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

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

[22] S. Vaid, C. Jones, H. Joho, and M. Sanderson. Spatio-textual indexing for geographical search on the web. *Advances in Spatial and Temporal Databases*, pages 923–923, 2005.

[23] D. Wu, M. Yiu, G. Cong, and C. Jensen. Joint top-k spatial keyword query processing. *TKDE*, 2011.

[24] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong. Efficient continuously moving top-k spatial keyword query processing. In *ICDE*, pages 541–552. IEEE, 2011.

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

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

[27] J. Zhang, X. Meng, X. Zhou, and D. Liu. Co-spatial searcher: Efficient tag-based collaborative spatial search on geo-social network. In *DASFAA*, pages 560–575, 2012.

[28] Y. Zhou, X. Xie, C. Wang, Y. Gong, and W. Y. Ma. Hybrid index structures for location-based web search. In *CIKM*, pages 155–162. ACM, 2005.