On Nearby-Fit Spatial Keyword Queries

Victor Junqiu Wei, Raymond Chi-Wing Wong, Cheng Long, and Pan Hui

Abstract—Geo-textual data is ubiquitous nowadays, where each object has a location and is associated with some keywords. Many types of queries based on geo-textual data, termed as *spatial keyword queries*, have been proposed, and are to find optimal object(s) in terms of both its (their) location(s) and keywords. In this paper, we propose a new type of query called *nearby-fit spatial keyword query* (NSKQ), where an optimal object is defined based not only on the location and the keywords of the object itself, but also on *those of the objects nearby*. For example, in an application of finding a hotel, not only the location of a hotel but also the objects near the hotel (e.g., shopping malls, restaurants and bus stops nearby) might need to be taken into consideration. The query is proved to be NP-hard, and in order to perform the query efficiently, we developed two approximate algorithms with small constant approximation factors equal to 1.155 and 1.79. We conducted extensive experiments based on both real and synthetic datasets, which verified our algorithms.

Index Terms—spatial keyword queries, proximity queries, spatial database, location-based services

1 Introduction

Nowadays, geo-textual data which refers to data with both spatial and textual information is ubiquitous. Some examples of geo-textual data include the spatial points of interest with textual description, geo-tagged web objects (e.g., webpages and photos at Flicker), and also geo-social networking data (e.g., users of FourSquare have their check-in histories, which are spatial, and their profiles, which are textual). The geo-textual data could be represented by a set of spatial objects each of which is associated with a set of (textual) keywords.

For example, Figure 1 shows our motivating example with a piece of geo-textual data. There are five types of objects distinguished by shapes, where each rectangular object is associated with keyword "hotel" meaning that it is a hotel, and similar rules apply to other objects. Note that each round object contains two keywords (i.e., "hotel" and "café"). Some distances are shown in the figure. Note that the distance between two objects is the distance between their centers.

Consider the following problem based on the geo-textual data presented in Figure 1.

A hotel-finding problem: Tom is going to attend an international conference and would like to find a hotel such that (1) the location of the hotel is near to the conference venue (marked by the black dot in the figure) and (2) there are at least a convenience store, a café and a pub nearby the hotel.

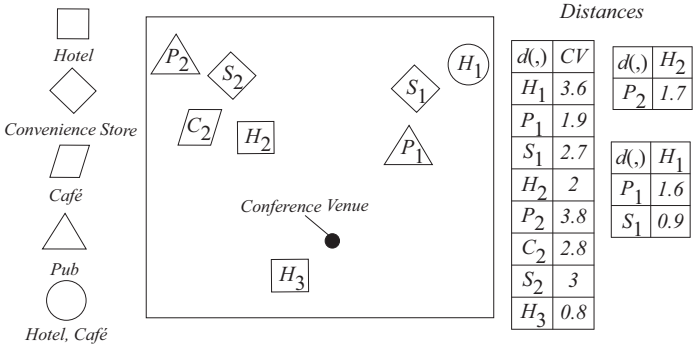


Fig. 1: A motivating example (CV stands for "Conference Venue")

It is easy to see that H_2 is the best selection for Tom since it is surrounded by a pub (P_2), a café (C_2) and a convenience store (S_2) and is in the vicinity of the conference venue. Besides, H_1 and H_3 are bad choices since there are nothing in the vicinity of H_3 and H_1 is far away from the conference venue.

Although many different types of queries based on geo-textual data which are termed as *spatial keyword queries* have been proposed in the literature, none of them could be used to capture the above hotel-finding problem well. We explain in detail by classifying these queries into four categories. The first category is the *spatial keyword kNN query* (SK*k*NNQ) [1], [2], [3], [4] which takes a query location and some query keywords as inputs and returns k closest objects to the query location among all objects each of which covers all query keywords. One attempt of using SK1NNQ to solve the problem is to specify the query location as the conference venue and the query keyword as "hotel", and the closest hotel to the conference venue, i.e., H_3, would be returned. The problem with this attempt is that it only captures the first criterion of the hotel-finding problem, but not the second one. An alternative attempt is to specify

- *V.J. Wei is with Noah's Ark Lab of Huawei Technologies. Major work was done when V.J.Wei was a Phd Student in the Department of Computer Science and Technology, Hong Kong University of Science and Technology. E-mail: wei.junqiu1@huawei.com*
- *R.C. Wong is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. E-mail: raywong@cse.ust.hk*
- *C. Long is with School of Computer Science and Engineering, Nanyang Technological University. Partial work was done when C. Long was a lecturer at Queen's University Belfast. E-mail: c.long@ntu.edu.sg*
- *P. Hui is with the Department of Computer Science at the University of Helsinki and the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. E-mail: panhui@cse.ust.hk*

the query location as the conference venue and the query keywords as "hotel", "convenience store", "café" and "pub", and no objects would be returned in this case since there exists no object which covers all these keywords. The drawback of this attempt is that it requires that all keywords are covered by a *single* object which is often not possible when there are multiple query keywords.

The second category is the *collective spatial keyword query* (CoSKQ) [5], [6], [7] which takes a query location and some query keywords as inputs and returns *a set of* objects such that the objects in the set *collectively* cover the query keywords and the distance from the set to the query location is the smallest. One attempt of using the CoSKQ query to solve the problem is to specify the query location as the conference venue and the query keywords as "hotel", "convenience store", "café" and "pub". No matter which distance function of CoSKQ is adopted and no matter what the parameters of CoSKQ are equal to, $\{H_1, S_1, P_1\}$ would be returned by a CoSKQ query, i.e., the best hotel H_2 is missed (see [8] for the detailed explanation). Thus, CoSKQ fails to capture the hotel-finding problem, and the key reason is that CoSKQ is not aware of the special role of 'hotel' in this problem which is Tom's target and the hotel in its solution could be far away from the conference venue as shown in the example.

The third category is *m Closest Keyword query* (*m*CK) [9], [10], [11]. The *m*CK query takes a set of m keywords as input and finds a set of m objects with the smallest diameter that cover the m keywords specified in the query where the diameter of a set is equal to the maximum pairwise distance between any two objects in this set. One attempt of using *m*CK to solve the problem is to specify the query keywords as "hotel", "convenience store", "café" and "pub", and H_1 (together with S_1 and P_1) would be returned. But, H_1 is far away from the conference venue.

The fourth category includes miscellaneous spatial keyword queries [12], [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26], [27], [28], [29], [30], [31], [32], [33], [34], [35], [36], [37], [38], [39], [40], [41] whose objectives are very different from that of the hotel-finding problem. Details could be found in Section 2.2.

Motivated by this, in this paper, we formalize the hotel-finding problem as a query called *nearby-fit spatial keyword query* (NSKQ) which has its query input q consisting of three parts: (1) a location $q.\lambda$ called the *query location*, (2) a keyword $q.t$ called the *target keyword* and (3) a set $q.\psi$ of keywords called *nearby keywords* and returns the object which has the *smallest* cost among all objects covering the target keyword, where the cost of an object is defined to be a linear combination of two parts: (1) the distance from the query location to the object and (2) the *smallest* distance from a set of objects collectively covering the nearby keywords to the object, where the distance from a set of objects to an object can be computed based on the *Diameter* distance function [6]. For example, the hotel-finding problem could be expressed by an NSKQ where the query location $q.\lambda$ is the conference venue, the target keyword $q.t$ is "hotel", and the set $q.\psi$ of nearby keywords is {"convenience store", "café", "pub"}. As could be verified, the answer to this NSKQ instance is H_2.

The key feature of NSKQ is that it evaluates each object not only by its own location and its keyword(s) but also its nearby objects. In other words, it accepts two types of keywords, namely a target keyword and some nearby keywords, which are used in two different ways. The target keyword is used to express users' requirements of what object to be found, called the *target object*, while nearby keywords are used to express users' requirements of what objects are preferred to be located nearby the target object. As a benefit of this distinguishable feature, NSKQ could be used to capture many queries based on geo-textual data where users' requirements/needs could be expressed in a fine-grained way. There are many other applications of NSKQ. We list two as follows.

- `house-finding problem`: it finds a house such that the house is near the work office and there are at least a restaurant, a supermarket and a swimming pool near the house, where the query location is the work office, the target keyword is "house" and nearby keywords are "restaurant", "supermarket" and "swimming pool".
- `geo-location based game`: in many geo-location based games such as Pokemon Go and InGress, each spatial object has some features which are essentially keywords (e.g. in Pokemon Go, each location is associated with some monsters). Consider Pokemon Go. In this game, each player collects monsters in some locations and then gathers at the so-called 'assembly place' (i.e., a POI) to play the monsters with each other. Suppose that Tom would like to find an assembly place to play games with other players where there are some desired monsters nearby so that he could by the way capture them. This query could be expressed in NSKQ where the query location is Tom's current location and the target keyword is 'assembly place' and the nearby keywords are the names of his desired monsters.

In many applications such as geo-location based game, one strong requirement is the real-time response. Unfortunately, NSKQ problem turns out to be NP-hard. A straightforward method for solving NSKQ is to compute the cost of each object covering the target keyword and return the one with the smallest cost. However, the problem of computing the cost of an object is intractable since the problem of computing the smallest distance from a set of objects covering the set of nearby keywords to a location is NP-hard [5], [6]. Therefore, in the case that there exist many objects covering the target keyword, the above algorithm is prohibitively expensive. An alternative idea is to design a branch-and-bound algorithm (like the algorithms in [5], [6] originally designed for other spatial queries) to find an optimal solution of the NSKQ problem. Still, as will be shown in our experiments, they are not efficient enough. Therefore, in this paper, we aim at developing fast approximate algorithms for NSKQ. Specifically, we developed two approximate algorithms, namely *Appro1* and *Appro2*, for NSKQ. *Appro1* provides a near-to-optimal approximation guarantee (i.e., 1.155-factor) and *Appro2* provides a slightly larger approximation factor (i.e., 1.79-factor) but with a better time complexity. Besides, our experiments show that they return near-to-optimal solutions in practice.

In summary, our main contributions are as follows. Firstly, we propose a new query called **nearby-fit spatial keyword query** (NSKQ) which has better expressiveness

than existing spatial keyword queries. Secondly, we prove that the NSKQ problem is NP-hard, and to solve it efficiently, we developed two approximate algorithms, both with small constant approximation factors. Thirdly, we conducted extensive experiments based on both real and synthetic datasets and verified the efficiency and the effectiveness of the approximate algorithms.

The rest of the paper is organized as follows. Section 2 formulates the NSKQ problem and reviews the related work, and Section 3 presents the two approximate algorithms for NSKQ. Section 4 gives the empirical study and Section 5 concludes the paper.

2 Problem Definition and Related Work

2.1 Problem Definition

Let D be a set of spatial objects each of which is associated with a set of keywords. Let o be an object in D. We denote by $o.\lambda$ the location of o and $o.\psi$ the set of keywords associated with o. Given a set O of objects and a set ψ of keywords, O is said to *cover* ψ if for each keyword in ψ, there is an object in O which contains the keyword, i.e., $\psi \subseteq (\cup_{o \in O} o.\psi)$. Given two objects o_1 and o_2 in D, we simply denote the distance between o_1 and o_2 by $d(o_1, o_2)$.

Given an object o and a set O of objects in D, we define $Dist(o, O)$ to be the distance from a set O of objects to an object o. In this paper, we adopt the Diameter function since this function is used in a lot of related studies like [5], [6], [11], [36], [37]. Specifically, given an object o and a set $O \subseteq D$ of objects, $Dist(o, O)$ denotes the diameter, i.e., the maximum pairwise distance, of the set $O \cup \{o\}$, i.e.,

$$Dist(o, O) = \max_{o_1, o_2 \in O \cup \{o\}} d(o_1, o_2) \quad (1)$$

Problem statement. Given a query q which consists of a location $q.\lambda$ called the *query location*, a keyword $q.t$ called the *target keyword*, and a set $q.\psi$ of keywords called the *nearby keywords*, the **nearby-fit spatial keyword query** (NSKQ) is to find the object in D which has the *smallest* cost wrt q among all objects containing the target keyword, where the cost of object o wrt q, denoted by $Cost(o|q)$, is defined to be a linear combination of two components: (1) the distance from the query location to o (i.e., $d(q, o)$) and (2) the *smallest* distance from a set of objects covering the set of nearby keywords to o (i.e., $\min\{Dist(o, O) | O \subseteq D, O \text{ covers } q.\psi\}$), i.e.,

$$Cost(o|q) = \alpha \cdot d(q, o) + (1-\alpha) \cdot \min\{Dist(o, O) | O \subseteq D, O \text{ covers } q.\psi\} \quad (2)$$

where $\alpha \in [0, 1]$ is a tuning parameter.

Besides, we safely assume that $q.t \notin q.\psi$ (since otherwise, we consider another query q' with $q'.\lambda = q.\lambda, q'.t = q.t$ and $q'.\psi = q.\psi \backslash \{q.t\}$, which is equivalent to q). We also assume that no object in D contains all nearby keywords and the target keyword (since otherwise, the solution could be found easily by scanning all objects in D and checking whether each object contains all nearby keywords and the target keyword).

Intractability. Unfortunately, NSKQ is NP-hard. We present this result in the following lemma.

Lemma 1. *NSKQ is NP-hard.*

Proof. We first give the decision problem of the NSKQ problem. Given a set D of spatial objects each of which is associated with a set of keywords, a query q consisting of a query location $q.\lambda$, a target keyword $q.t$ and a set $q.\psi$ of nearby keywords, and a non-negative real value r, the problem is to determine whether there exists an object $o \in D$ such that $Cost(o|q)$ is at most r.

We prove by transforming the *CoSKQ* problem [6] which is known to be NP-hard to our NSKQ problem.

The decision problem of CoSKQ is given as follows. Given a set D' of spatial objects each of which is associated with a set of keywords, a query q' consisting of a query location $q'.\lambda$ and a set $q'.\psi$ of query keywords, and a non-negative real value r', the problem is to determine whether there exists a set of objects $O \subseteq D'$ such that O covers $q'.\psi$ and $Dist(q', O)$ is at most r'.

We then construct an NSKQ problem instance based on a CoSKQ problem instance as follows. We create a dummy keyword t which is different from any existing keyword of an object in D'. We construct D as $D' \cup \{o'\}$ where o' is a dummy object with $o'.\lambda = q'.\lambda$ and $o'.\psi = \{t\}$. We construct q where $q.\lambda = q'.\lambda$, $q.t = t$, and $q.\psi = q'.\psi$. We set r to r'. It is easy to verify that the above construction could be done in polynomial time.

It could be verified that the CoSKQ problem instance and the constructed NSKQ problem instance are equivalent since o' is the only object in D covering the target keyword t and in the NSKQ problem instance, $Cost(o'|q) = \alpha \cdot d(q, o') + (1 - \alpha) \cdot \min\{Dist(o', O) | O \subseteq D, O \text{ covers } q.\psi\} = \alpha \cdot 0 + (1 - \alpha) \cdot \min\{Dist(q', O) | O \subseteq D', O \text{ covers } q'.\psi\}$, which is the $(1 - \alpha) Dist(\cdot, \cdot)$ cost in the CoSKQ instance. ■

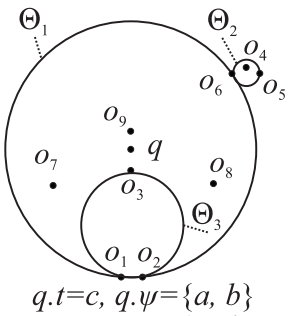


Fig. 2: An example of NSKQ

Obj	o_1	o_2	o_3	o_4	o_5	o_6	o_7	o_8	o_9
ψ	a	b	c	a	b, c	b, c	a, c	b, c	c

TABLE 1: Keyword set of each object

Notations: In the following, we define some notations which will be used later in the paper. An object o_p is said to be a *target object* if it contains $q.t$. An object o_p is said to be the *optimal target object* if (1) it is a target object and (2) it has the smallest value of $Cost(o_p|q)$ among all target objects (i.e., $Cost(o_p|q) = \min\{Cost(o|q) | o \text{ is a target object}\}$). Given a set O of objects and a target object o_p, O is said to be a *nearby set* of o_p if O covers $q.\psi$. Given a set O of objects and a target object o_p, O is said to be the *best nearby set* of o_p if (1) O is a nearby set of o_p and (2) it has the smallest value of $Dist(o_p, O)$ among all possible nearby sets of o_p (i.e., $Dist(o_p, O) = \min\{Dist(o_p, O') | O' \text{ is a nearby set of } o_p\}$). Given a target object o and its nearby set O, we define $Cost(o|q, O)$ to be $\alpha \cdot d(q, o) + (1 - \alpha) \cdot Dist(o, O)$.

Figure 2 shows an example of NSKQ. There are 9 objects and 1 query q. The keyword set of each object is presented

in Table 1. In this example, $q.t = c$ and $q.\psi = \{a, b\}$. o_3 is the optimal target object and $\{o_1, o_2\}$ is the best nearby set of o_3.

It is worth mentioning that although the original problem setting of NSKQ considers a *single* target keyword for the ease of illustration, any algorithm originally designed for handling a single target keyword could also handle the case when an object to be returned covers *multiple* target keywords (by replacing the operation of selecting objects covering a single target keyword with the operation of selecting objects covering multiple target keywords).

2.2 Related Work

Many types of queries have been proposed based on geo-textual data, which we classify into four categories.

The first three categories are *spatial keyword kNN query* (SK*k*NNQ) [1], [2], [3], [4], *collective spatial keyword query* (CoSKQ) [5], [6], [7], [11] and *m Closest Keyword query* (*m*CK) [9], [10], [11]. We refer the readers to Section 1 for the details due to the limited space.

The fourth category includes some miscellaneous spatial keyword queries, whose objectives are very different from NSKQ, as explained below. The spatial keyword reverse top-k query [12], [13], [14], [15] finds a set of objects whose spatial keyword kNN query results include the query location. The *spatial keyword top-k query* [16], [17], [18] is to take a query location and a set of keywords as inputs and return k objects with the greatest scores computed based on both their locations and their relevance to the query keywords. The *continuous spatial keyword query* [19], [20], [21] studies a spatial keyword query based on *moving objects*. The *spatial keyword why-not query* [22], [23] is a spatial keyword kNN query enhanced with the feedback from users. The feedback is given in the form of several desired objects which could not be found in the initial output and the feedback is used to refine the query processing to obtain a final satisfied output. The *spatial keyword query over data streams* [24], [25], [26], [27], [28], [29], [30] is under the steaming model where data is continuously generated. The *direction-aware spatial keyword query* [31] is a spatial keyword kNN query with the consideration of the direction constraint. The *spatial-textual similarity join* problem [32], [33], [34], [35] finds all "similar" pairs of objects from a set (or two sets) of geo-tagged objects where the similarity between two objects is computed based on the spatial closeness and the textual similarity. The *best keyword cover search* [36] assumes that each object has a location, a set of keywords and a rating score, and finds a set O of objects covering a set of query keywords with the greatest score computed based on the diameter of this set and their rating scores. *SK-Cover* [37] finds a set O of objects covering a set of query keywords such that they have the smallest $(|O| - 1) \cdot \max_{o_1, o_2 \in O} d(o_1, o_2)$. In the *clue-based spatio-textual query* [38], a query consists of (1) the *category* (or called the target keyword in our paper) of the object o_p to be found and (2) a number of *clue objects*, each of which contains a number of keywords and its relative position w.r.t. o_p (represented by the direction and the distance). It finds the object o_p with the category specified in the query such that for each clue object o_c, there exists an object o in D, where $o_c.\psi \subseteq o.\psi$ and $o_c.\lambda \approx o.\lambda$ (note that $o_c.\lambda$ could be computed from $o_p.\lambda$ and the relative position information of o_c given in the query). This query is usually used when a user went to visit the object o_p before but after a certain of time, s/he would like to search for o_p again with a *rough* memory about the information of some clue objects related to o_p (e.g., keywords and distance). Thus, this query requires the user to provide the distance information (for clue objects) which is not required by NSKQ. Rocha-Junior, etc. [40] proposed the Spatial Preference Query (SPQ) which ranks each object by the feature objects in its vicinity, where the feature object is an object belonging to one of the user-specified categories. Although each category could be treated as a keyword, there are several other major differences between SPQ and NSKQ. First, the vicinity of an object utilized in SPQ must be a circle or a star-shaped graph both centered at the object (i.e., a fixed distribution/shape). Note that other nodes in the star-shaped graph are nearest neighbors of the object in each category. However, in NSKQ, there is no priori knowledge of the distribution of the nearby objects of an object. Secondly, the ranking function of SPQ is an aggregate function of the ratings of the object's feature objects. Thirdly, SPQ does not have query location. But, in NSKQ, the objective function is an aggregate distance function of the object and its nearby objects. Thus, in this sense, SPQ and NSKQ have different goals. Later, Li, etc. [39] proposed Location-Aware Group Preference (LGP). LGP considered multiple users and ranks each object by the the combination of the following two components: 1. the sum of the distances between the object and each user and 2. the sum of the distances between the object and its nearest neighbors in each category. Again, the vicinity adopted in LGP is a star-shaped graph centered at the object with its nearest neighbors in each category as other nodes. Due to the strong priori assumption of SPQ and LPQ imposed on the vicinity, their techniques could not be applied to NSKQ. [41] studies the spatial keyword query problem in a setting where each object is associated with an inherent cost (e.g., workers have monetary costs).

Many index structures have been developed for efficient spatial keyword queries. Some examples include *IR-tree* [16], *bR-tree*, IR^2*-tree* [9] and I^3 index [42]. Chen, etc. [43] conducted a comprehensive empirical study on the existing index structures of the spatial keyword query.

2.3 Adaptions of Algorithms for Existing Spatial Keyword Queries

Although the existing spatial keyword queries are different from NSKQ, the algorithms for some of these existing queries could be adapted to solve NSKQ but they either provide a not-that-good approximation factor (e.g., ≥ 2) or has a high time complexity (e.g., worse than $O(|D|^2)$). Specifically, we explored the adaptions of the algorithms for CoSKQ and *m*CK (specifically, the approximate algorithms for CoSKQ and *m*CK since both problems are NP-hard and the exact algorithms are not scalable) and show that they could in fact provide some constant approximation factors, which we would introduce next (we did not consider the adaptions of the algorithms for other existing queries since they cannot provide any reasonable approximation guarantees).

Adaptions of the Existing Algorithms of CoSKQ. In the literature, three approximate algorithms have been developed

for CoSKQ, namely *Cao-Appro1* [5], *Cao-Appro2* [5] and *Long-Appro* [6]. Let A be any of these three algorithms. We have two options of adapting A for NSKQ.

Option 1. We adapt algorithm A for an NSKQ query q with two steps. First, for each object o containing the target keyword, we execute algorithm A for a CoSKQ query with the query location as $o.\lambda$ and the set of query keywords as $q.\psi$ and obtain a set O of objects that cover $q.\psi$. Second, among all these o's, we return the one with $Cost(o|q,O)$ the smallest where O is the corresponding set of objects returned by algorithm A for the object o. Note that in this adaption, algorithm A is executed $|D_{q.t}|$ times where $D_{q.t}$ denotes the set containing all target objects.

The algorithms of adapting Cao-Appro1, Cao-Appro2 and Long-Appro with the above Option 1 are called *Cao-Appro1-Adapt1*, *Cao-Appro2-Adapt1*, and *Long-Appro-Adapt1*, respectively.

Option 2. According to [5], [6], during the process of algorithm A for a CoSKQ query, many sets of objects that cover the set of query keywords are constructed, and among these sets, the one with smallest distance to the query location is returned.

We adapt algorithm A for an NSKQ query q with two steps. First, we execute algorithm A for a CoSKQ query with the query location as $q.\lambda$ and the set of query keywords as $q.\psi \cup \{q.t\}$, and during the execution process, whenever a set O is constructed, we pick the nearest object o in O from $q.\lambda$ that covers the target keyword $q.t$ (note that such o always exist since $q.t$ is a query keyword for the CoSKQ query). Second, among all these o's, we return the one with $Cost(o|q,O)$ the smallest, where O is the corresponding set of objects. Note that in this adaption, algorithm A is executed only once.

Similarly, the algorithms of adapting Cao-Appro1, Cao-Appro2 and Long-Appro with the above Option 2 are called *Cao-Appro1-Adapt2*, *Cao-Appro2-Adapt2*, and *Long-Appro-Adapt2*, respectively.

Adaptions of the Existing Algorithms of mCK. In the literature, two approximate algorithms have been developed for mCK, namely *SKEC* [11] and *SKEC+* [11]. Let A be any of these two algorithms. According to [11], during the process of algorithm A for an mCK query, many sets of objects that cover the set of query keywords of the query are constructed, and among these sets, the one with smallest diameter is returned.

We adapt A for an NSKQ query q with two steps. First, we execute algorithm A for an mCK query with the set of query keywords as $q.\psi \cup \{q.t\}$, and during the execution process, whenever a set O is constructed, we pick the nearest object o in O from $q.\lambda$ that covers the target keyword $q.t$ (note that such o always exists since $q.t$ is a query keyword for the mCK query). Second, among all these o's, we return the one with $Cost(o|q,O)$ the smallest, where O is the corresponding set of objects. Note that in this adaption, algorithm A is executed only once.

Similarly, the algorithms of adapting *SKEC* and *SKEC+* are called *SKEC-Adapt* and *SKEC+-Adapt*, respectively. Since *SKEC* and *SKEC+* were originally designed for *mCK* which has no query location, in *SKEC-Adapt* and *SKEC+-Adapt*, we also provide a heuristic-based pruning method to make them query-location-aware. This pruning method works as a prelude of *SKEC-Adapt* and *SKEC+-Adapt* and it prunes a large amount of objects outside the vicinity of the query location. Specifically, the pruning method first executes *Cao-Appro2-Adapt2*. Let o' and O' denote the target object and its nearby set found by *Cao-Appro2-Adapt2*. Then, in the execution of *SKEC-Adapt* and *SKEC+-Adapt*, all objects whose distances to the query location are greater than $C' = \frac{Cost(o'|q,O')}{\min\{\alpha,1-\alpha\}}$ are omitted. The correctness of this pruning is based on the following lemma.

Lemma 2. *Consider any target object o_t and any nearby set O of o_t. $Cost(o_t|q,O) > Cost(o'|q,O')$ if the distance between an object o and q is larger than C' where $o \in O \cup \{o_t\}$.*

Proof. For the sake of limited space, the proof could be found in [8]. ■

Thus, there is no need to consider any object $o \in O \cup \{o_t\}$ whose distance to q is larger than C' since it gives a solution worse than o', where o_t is an target object and O is a nearby set of o_t.

Although BKC [36] is a generalized version of mCK (i.e., BKC reduces to mCK when $\alpha = 1$ in its objective function and thus is NP-hard.), we do not consider the adaption of the BKC algorithm to NSKQ since the only algorithm proposed in [36] is an exact algorithm which is not scalable (it takes more than 200s on a dataset with only 678,581 POIs and 187 unique keywords when there are more than 6 query keywords).

Summarization of the Adapted Algorithms. We analyze the approximation factors and also time complexities of the above adapted algorithms and present them in Table 2. We refer the readers to [8] for the analysis. As mentioned before, it is usually the case that an algorithm either provides a not-that-good approximation factor (e.g., ≥ 2) or has a high time complexity (e.g., worse than $O(|D|^2)$).

In this paper, we develop two new algorithms, namely *Appro1* and *Appro2* for NSKQ, which provide better trade-offs between the approximation factor and the running time compared with the aforementioned adapted algorithms.

3 Processing NSKQ

As shown in Lemma 1, NSKQ is NP-hard. The running time of any exact algorithm designed for NSKQ is large. In the experimental results (to be shown later), we will show that an adapted exact algorithm for NSKQ is very slow. Thus, we propose two approximate algorithms for NSKQ, namely *Appro1* and *Appro2*, which are much more efficient, in Section 3.1 and Section 3.2, respectively. We assume that an indexing structure is built on the dataset and the indexing structure supports the spatial keyword kNN query and the spatial circular range query which finds all objects in a given disk. The indexing structure could be but not limited to the IR-tree [16] which supports the aforementioned queries.

3.1 Approximate Algorithm 1

In this section, we present an algorithm, *Appro1*, based on a concept of "minimal covering disk", which has its approximate ratio of 1.155 and its time complexity of $O(|q.\psi|^3 \cdot |D| \log |D|)$.

Algorithm	Approx. Factor for NSKQ	Time Complexity	Small Approximate Factor (< 2)?	Low Time Complexity (Lower Than $|D|^2$)?
Cao-Appro1-Adapt1 [5]	2	$O(\|q.\psi\||D|\log|D|)$	no	yes
Cao-Appro2-Adapt1 [5]	2	$O(\|q.\psi\||D|^2\log|D|)$	no	no
Long-Appro-Adapt1 [6]	1.73	$O(\|q.\psi\||D|^2\log|D|)$	yes	no
Cao-Appro1-Adapt2 [5]	$\max\{2, \frac{2}{\alpha}-1\}$ (at least 2)	$O(\|q.\psi\|\log|D|)$	no	yes
Cao-Appro2-Adapt2 [5]	2	$O(\|q.\psi\||D|\log|D|)$	no	yes
Long-Appro-Adapt2 [6]	$\max\{\frac{2-\alpha}{1-\alpha}, \frac{2-\alpha}{\alpha}\}$ (at least 2)	$O(\|q.\psi\||D|\log|D|)$	no	yes
SKEC-Adapt [11]	1.155	$O(\|D\|^4)$	yes	no
SKEC+-Adapt [11]	1.155+ϵ	$O(\|q.\psi\||D|^2\log|D|\log\frac{1}{\epsilon})$	yes	no
Appro1 (i.e., our algorithm)	1.155	$O(\|q.\psi\|^3\cdot|D|\log|D|)$	yes	yes
Appro2 (i.e., our algorithm)	1.79	$O(\|q.\psi\||D|\log|D|+|D|\log^2|D|)$	yes	yes

TABLE 2: Comparison of Algorithms where $|D|$ is large and $|q.\psi|$ is small in practice

The major idea of *Appro1* is described as follows. Roughly speaking, "minimal covering disk" is a disk *compactly* covering all the nearby keywords and the target keyword. Due to this property, we propose a *preliminary* version of *Appro1* which (1) first enumerates all possible *candidates* of minimal covering disks, (2) for each enumerated disk, finds the nearest object o'_p from query q in the disk containing the target keyword and a set O' of objects in the disk covering all nearby keywords, and computes the cost value $Cost(o'_p|q, O')$, and (3) returns the object o'_p with the smallest cost. Later, we show that this preliminary version is a 1.155-approximate algorithm. However, there is a major challenge in this preliminary version which is how to enumerate all possible candidates of minimal covering disks. As discussed later, a straightforward implementation is very expensive. Motivated by this, based on an interesting observation called the *corner-edge observation* to be introduced later, we propose a *complete* version of *Appro1* with an efficient implementation of enumerating all possible candidates of minimal covering disks. The approximate ratio of this complete version is still 1.155 and its time complexity is $O(|q.\psi|^3\cdot|D|\log|D|)$.

We first give some important concepts used in our approximate algorithm, including the concept of "minimal covering disk", in Section 3.1.1. Then, we give our proposed algorithm called *Appro1* in Section 3.1.2.

3.1.1 Notations

Firstly, we define the concept of "minimal covering disk" as follows. Given a disk Θ, Θ is said to be a *covering disk* if there exists a set O of objects in Θ such that O covers all nearby keywords and the target keyword. Given a disk Θ, Θ is said to be a *minimal covering disk* if Θ is a covering disk and there does not exist a smaller disk Θ' such that Θ' is a covering disk and Θ' does not contain any object outside Θ. Given a point q and a non-negative real number r, we denote the disk centered at q with radius equal to r by $\mathbb{D}(q, r)$.

For example, Figure 2 shows 9 objects, namely $o_1, o_2, ..., o_9$ as well as three disks, namely Θ_1, Θ_2 and Θ_3, where $\Theta_1 = \mathbb{D}(q, d(q, o_6))$, Θ_2 is a disk whose boundary contains o_5 and o_6, and Θ_3 is a disk whose boundary contains o_1, o_2 and o_3. Consider that $q.t = c$ and $q.\psi = \{a, b\}$. Θ_1, Θ_2 and Θ_3 are covering disks since there exists a set O of objects in each of these disks such that O covers all nearby keywords and the target keyword. Neither Θ_1 nor Θ_2 is a minimal covering disk (since there exists a smaller covering disk of Θ_1 and there exists a smaller covering disk of Θ_2) but Θ_3 is a minimal covering disk.

We defined the concept of "minimal covering disk". Next, we present two observations about "minimal covering disk" (which will be used to generate "candidates" of "minimal covering disk" in our algorithm, *Appro1*). Before that, we define more concepts which will be used in the observations.

Given an object o in D and a disk Θ, o is said to be a *unique object* in Θ if o has a keyword t in $q.\psi \cup \{q.t\}$ and no other objects in Θ excluding the boundary of Θ have t. For example, consider Θ_2 in Figure 2. Each of the three objects in Θ_2 is a unique object in Θ_2 since it has a keyword t in $q.\psi \cup \{q.t\}$ and no other objects in Θ_2 excluding the boundary of Θ_2 have t. Note that although $o_5.\psi = o_6.\psi(= \{b, c\})$, o_5 and o_6 (on the boundary of Θ_2) are both unique objects in Θ_2.

Given an object o in D and a disk Θ, o is said to be a *unique boundary object* of Θ if o is along the boundary of Θ and o is a unique object in Θ. For example, consider Θ_2 in Figure 2. o_5 and o_6 are both unique boundary objects of Θ_2 because each of them is a unique object in Θ_2 and is along the boundary of Θ_2. But, o_4 is not a unique boundary object of Θ_2 because it is not along the boundary of Θ_2.

It is well known that a disk could be determined by three points on its boundary or two points on its boundary lying on a line passing through its center. Equipped with the above notations and concepts, we are ready to present the observations.

Observation 1. *Given a disk Θ, if Θ is a minimal covering disk, then there exist two or three unique boundary objects of Θ. If there are exactly two unique boundary objects of Θ, then the two objects must lie on a line passing through the center of Θ.*

Proof. The proof could be found in [8]. ■

Consider our running example in Figure 2. Θ_2 is the case for two objects. Disk Θ_3 is the case for three objects.

According to the above observation, we know that there are two possible cases. The first case is that there exist exactly two unique boundary objects of disk Θ. We call this case the *two-object covering case*. The second case is that there exist three unique boundary objects of Θ. We call this case the *three-object covering case*.

Note that in the two-object covering case, we know that there exist exactly two unique boundary objects of Θ. In this case, the minimal covering disk is a disk whose center is equal to the mid-point of these two objects and whose radius is equal to the distance between the mid-point and one of these two objects.

It is worth mentioning that any more-than-three-object covering case belongs to three-object covering case. But a three-object covering case may be or may not be a two-object

covering case since the later has the requirement on the positions of its unique boundary objects while the former does not have such a requirement.

Next, we give our next observation.

Observation 2. *Given a covering disk Θ, if there exists two or three unique boundary objects of Θ, then Θ can be a minimal covering disk or not.*

For example, consider Θ_3 in Figure 2. It has three unique boundary objects o_1, o_2, o_3 and it is a minimal covering disk. Consider Θ_2 in Figure 2. It has two unique boundary objects o_5 and o_6 but it is not a minimal covering disk.

Given a disk Θ, we say that Θ is a *candidate* of a minimal covering disk if Θ is a covering disk and one of the following conditions holds: (1) there exist exactly two unique boundary objects lying on a line passing through the center of Θ and (2) there exist three unique boundary objects of Θ.

We want to emphasize that based on Observation 1, if disk Θ is a minimal covering disk, then Θ is a candidate of a minimal covering disk. However, we know that based on Observation 2, if disk Θ is a candidate of a minimal covering disk, then Θ can be a minimal covering disk or not. In conclusion, the set of all possible candidates of minimal covering disks is a proper superset of the set of all possible minimal covering disks.

3.1.2 Algorithm

Based on the concept of "minimal covering disk", one may come up with the following three-step method called the preliminary version of *Appro1*.

- **Step 1 (Initialization):** A variable o_p, denoting the best-known target object, is initialized as $\emptyset$. A variable O, denoting the best-known nearby set of o_p, is initialized to $\emptyset$. A variable C, denoting the current best-known cost, is initialized to ∞.
- **Step 2 (Minimal Covering Disk Finding):** For each candidate of the minimal covering disk Θ, we do the following. We find the nearest object o'_p from query q in Θ containing the target keyword. We find a set O' of objects in Θ covering all nearby keywords. (How to find objects in Θ covering all nearby keywords does not affect the approximate ratio of *Appro1*. We will present how we execute this step later.) If $Cost(o'_p|q, O') < C$, then we update O with O', o_p with o'_p, and C with $Cost(o'_p|q, O')$.
- **Step 3 (Output):** We return o_p as the output.

The following theorem shows the approximate factor of the preliminary version of *Appro1*.

Theorem 1. *The preliminary version of* Appro1 *returns a 1.155-approximate solution for NSKQ.*

Proof: The proof can be found in [8]

It is interesting to know that the preliminary version of *Appro1* is a 1.155-approximate algorithm but as described before, the remaining issue is how to enumerate all possible candidates of minimal covering disks efficiently in Step 2. After we address this issue, we have a *complete* version of *Appro1*.

Straightforward Implementation of Enumerating All Possible Candidates: In the following, we first present a straightforward implementation of enumerating all possible candidates of minimal covering disks which is very costly. Then, we propose a novel and efficient implementation.

The straightforward implementation includes the following two sub-steps for finding candidates of the minimal covering disks based on Observation 1.

- **Step (a) (Disk Enumeration):** For each possible combination of three objects, say o_1, o_2 and o_3, in D, we construct a disk Θ whose boundary passes through these three objects. For each possible combination of two objects, say o_1 and o_2, in D, we construct a disk Θ whose boundary passes through these two objects (i.e., a disk whose center is equal to the mid-point of these two objects and whose radius is equal to the distance between the mid-point and one of these two objects).
- **Step (b) (Minimal Covering Disk Candidate Verification):** For each disk Θ constructed in Step (a), we check whether Θ is a candidate of a minimal covering disk by checking the following two conditions: (1) whether Θ is a covering disk and (2) whether, for each object o of the objects involved in constructing Θ in Step (a), o is a unique boundary object of Θ. If both conditions are satisfied, Θ is a candidate of a minimal covering disk. Otherwise, Θ is not and in this case, we prune Θ.

Efficient Implementation of Enumerating All Possible Candidates: However, Step (a) of the straightforward implementation is costly because it has to enumerate all possible combinations of three objects in D and all possible combinations of two objects in D. In the following, based on a new observation called the *corner-edge observation* to be introduced later, we propose an efficient algorithm which considers all possible combinations of two/three "close" objects in D only for generating candidates of minimal covering disks and prune all possible combinations of two/three "distant" objects in D which are unnecessary because the disk constructed by each combination of these two/three "distant" objects is not a minimal covering disk (since it is not "compact" enough).

Before we introduce the corner-edge observation, we introduce some concepts first.

Given a Voronoi Diagram (VD) built on a set of points, a disk Θ is called a *corner disk* of VD if the center of Θ is at one of the corners of a cell of VD and the radius of Θ is equal to the distance between this corner and the object occupying this cell. A disk Θ is called an *edge disk* of VD if the center of Θ is at the mid-point between two objects occupying the two cells separated by an edge of VD and the radius of Θ is equal to the distance between this mid-point and one of these two objects.

Consider our running example as shown in Figure 2. Figure 3(a) shows the Voronoi diagram built on all objects. Note that point x is a corner of the cell occupied by object o_1. Disk Θ is a disk centered at the corner x with the radius equal to $d(x, o_1)$. Note that this radius is also equal to $d(x, o_2)$ and $d(x, o_3)$. Thus, Θ is a corner disk of this Voronoi Diagram. Figure 3(b) shows the Voronoi diagram built on all objects except o_2. Note that point y is the mid-point between o_4 and o_6 whose cells are separated by an edge of the Voronoi diagram. Disk Θ' is a disk centered at point y with the radius equal to $d(y, o_4)$. Note that this radius is

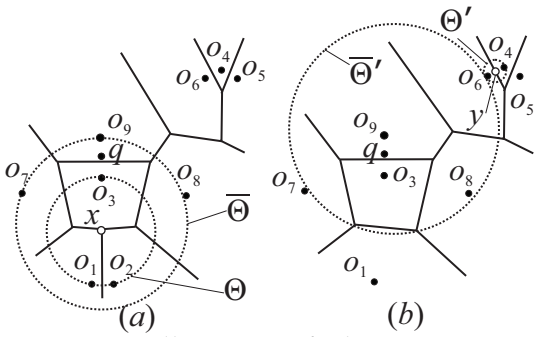


Fig. 3: Illustration of Observation 3

also equal to $d(y, o_6)$. Thus, Θ' is an edge disk of the Voronoi Diagram.

Let S_t be a set of objects in D containing keyword t. Consider three distinct keywords t_1, t_2 and t_3. Let $VD(t_1, t_2, t_3)$ be the Voronoi diagram built on all objects in $S_{t_1} \cup S_{t_2} \cup S_{t_3}$. Let $VD(t_1, t_2)$ be the Voronoi diagram built on all objects in $S_{t_1} \cup S_{t_2}$.

Observation 3 (Corner-Edge Observation)**.** *If Θ is a minimal covering disk, then one of the following two conditions holds.*

- *(1) There exist three distinct keywords, namely t_1, t_2 and t_3, in $q.\psi \cup \{q.t\}$ such that Θ is a corner disk of $VD(t_1, t_2, t_3)$,*
- *(2) There exist two distinct keywords, namely t_1 and t_2, in $q.\psi \cup \{q.t\}$ such that Θ is an edge disk of $VD(t_1, t_2)$.*

Proof. The proof could be found in [8] ■

Consider back our running example (Figure 2). Consider three distinct keywords, namely a, b and c. We know that $S_a \cup S_b \cup S_c$ is a set of all objects in the example (i.e., $o_1, o_2, ..., o_9$). Figure 3(a) shows the Voronoi diagram built on all objects in $S_a \cup S_b \cup S_c$ (i.e., $VD(a, b, c)$). Note that as described before, Θ is a minimal covering disk. In Figure 3(a), Θ is also a corner disk of $VD(a, b, c)$.

Consider two distinct keywords, namely a and c. We know that $S_a \cup S_c$ is a set of all objects except o_2. Figure 3(b) shows the Voronoi diagram built on all objects in $S_a \cup S_c$ (i.e., $VD(a, c)$). It is easy to verify that Θ' is a minimal covering disk. In Figure 3(b), Θ' is also an edge disk of $VD(a, c)$.

Note that disk $\overline{\Theta}$ (in Figure Figure 3(a)) whose boundary contains the objects o_7, o_8 and o_9 is not a corner disk of $VD(a, b, c)$ and an edge disk of the Voronoi diagram built on all objects in $S_t \cup S_{t'}$ for any two distinct keywords t and t' in $\{a, b, c\}$, Thus, it is not a minimal covering disk. Similarly, the disk $\overline{\Theta}'$ (in Figure 3(b)) centered at the mid-point between o_6 and o_7 with the radius equal to the distance between this mid-point and o_6 is not a corner disk of $VD(a, b, c)$ and an edge disk of the Voronoi diagram built on all objects in $S_t \cup S_{t'}$ for any two distinct keywords t and t' in $\{a, b, c\}$. Thus, it is not a minimal covering disk.

We want to emphasize that the straightforward implementation requires to enumerate disks like $\overline{\Theta}$ and $\overline{\Theta}'$ constructed by two/three "distant" objects but our efficient implementation to be introduced later does not.

With the corner-edge observation, we are ready to describe our efficient algorithm for Step (a) involving the following steps.

- **Step (i) (Keyword-Driven Set Finding):** For each distinct keyword t, we find a set S_t of objects in D containing t.
- **Step (ii) (Possible Candidate Disk Generation):** We perform the following two operations.
  - The first operation is shown as follows. If $|q.\psi \cup \{q.t\}| \geq 3$, for each possible triple of three distinct keywords, namely t_1, t_2 and t_3 from $q.\psi \cup \{q.t\}$, we do the following.
    - **Step (I):** We build a Voronoi diagram $VD(t_1, t_2, t_3)$
    - **Step (II):** We construct all possible corner disks of $VD(t_1, t_2, t_3)$. Note that there are $O(|D|)$ corner disks.
  - The second operation is shown as follows. For each possible pair of two distinct keywords, namely t_1 and t_2, from $q.\psi \cup \{q.t\}$, we do the following.
    - **Step (A):** We build a Voronoi diagram $VD(t_1, t_2)$
    - **Step (B):** We construct all possible edge disks of $VD(t_1, t_2)$. Note that there are $O(|D|)$ edge disks.

After we introduce the above two-step method, we have a complete version of *Appro1*. In the following, when we write *Appro1*, we mean the complete version of *Appro1*. The following lemma shows that the above two-step method (i.e., Step (i) and Step (ii)) does not affect the approximate ratio of *Appro1*.

Lemma 3. *The approximate ratio of* Appro1 *is still 1.155 even if the set of candidates of minimal covering disks considered by* Appro1 *is the set of all possible corner disks and all possible edge disks.*

Proof. The proof could be found in [8] ■

Detailed Steps and Time Complexity: *Appro1* involves three steps, namely Step 1, Step 2 and Step 3. Step 1 takes $O(1)$ time.

Consider Step 2. We first give the time complexities of the two fundamental operations used in Step 2 and then analyze the time complexity of Step 2. The first operation is the operation of finding a nearest neighbor containing a given keyword from a query point, whose time cost is denoted by T_{NN-Key}. Note that $T_{NN-Key} = O(\log |D|)$ [44] (since a nearest neighbor query could be issued on a Voronoi diagram built based on all objects containing a particular keyword). The second operation is the operation of finding a nearest neighbor containing a given keyword from a query point in a given disk, whose time cost is denoted by $T_{NN-Key-Disk}$. Note that $T_{NN-Key-Disk} = O(\log |D| + l)$ [45], where l is the number of objects in the given disk (since a range query could be issued for finding all objects in a disk and then a linear scan of all objects in a disk could be involved for finding the nearest neighbor from a query point).

Now, we are ready to analyze the time complexity of Step 2. In Step 2, we have to enumerate all possible candidates of minimal covering disks, which involves Step (a) and Step (b). Consider Step (a) which involves two steps, namely Step (i) and Step (ii). Step (i) takes $O(|D| \cdot |q.\psi|)$ time. Step (ii) includes two operation. Consider the first operation. The first operation executes Step (I) and Step (II). For each possible triple of three distinct keywords, namely t_1, t_2 and t_3, Step (I) takes $O(|D| \log |D|)$ time. Step (II) takes $O(|D|)$ time (since there are $O(|D|)$ corner disks). Since there are $\binom{|q.\psi|+1}{3}$ possible triples, the first operation takes $O(\binom{|q.\psi|}{3}) \cdot (|D| \log |D|))$

time. Consider the second operation. For each possible pair of two distinct keywords, namely t_1 and t_2, Step (A) takes $O(|D|\log|D|)$ time. Step (B) takes $O(|D|)$ time (since there are $O(|D|)$ edge disks). Since there are $\binom{|q.\psi|+1}{2}$ possible pairs, the second operation takes $O(\binom{|q.\psi|}{2}\cdot(|D|\log|D|))$ time. So, Step (ii) takes $O(\binom{|q.\psi|}{3}\cdot(|D|\log|D|))$ time. Thus, Step (a) takes $O(|D|\cdot|q.\psi|+\binom{|q.\psi|}{3}\cdot(|D|\log|D|))$ time. Note that there are $O(\binom{|q.\psi|}{3}\cdot|D|)$ possible candidates generated in Step (a).

Consider Step (b) which checks whether each disk satisfies two conditions: (1) whether the disk is a covering disk and (2) whether for each object o of the objects involved in constructing the disk in Step (a), o is a unique boundary object of the disk. Since each checking operation on a disk can be implemented by a range query from the center of the disk (which takes $O(\log|D|+l)$ time where l is the number of objects in the disk) and an operation of comparing whether each object involved in constructing the disk in Step (a) is a unique boundary object of the disk (which takes $O(l)$ time), Step (b) takes $O(\binom{|q.\psi|}{3}\cdot|D|(\log|D|+l))$ time.

Thus, Step (a) and Step (b) (for finding candidates of the minimal covering disks in Step 2) take $O(|D|\cdot|q.\psi|+\binom{|q.\psi|}{3}\cdot(|D|\log|D|)+\binom{|q.\psi|}{3}\cdot|D|(\log|D|+l)) = O(|D|\cdot|q.\psi|+\binom{|q.\psi|}{3}\cdot|D|(\log|D|+l))$ time. In Step 2, for each candidate of the minimal covering disk, we find the nearest object from the query in a disk containing the target keyword (which takes $O(\log|D|+l)$), find a set of objects in a disk covering all nearby keywords (by finding all objects in a disk with a range query and incrementally inserting objects found satisfying "uncovered" nearby keywords to a set variable, initialized to an empty set, until the set variable covers all nearby keywords) (which takes $O(\log|D|+l)$) and update the best-known variables (which takes $O(|q.\psi|)$). The overall time complexity of Step 2 is $O(|D|\cdot|q.\psi|+\binom{|q.\psi|}{3}\cdot|D|(\log|D|+l)+\binom{|q.\psi|}{3}\cdot|D|\cdot(\log|D|+l+\log|D|+l+|q.\psi|)) = O(|D|\cdot|q.\psi|+\binom{|q.\psi|}{3}\cdot|D|(\log|D|+l+|q.\psi|))$.

Step 3 takes $O(1)$ time.

The overall time complexity of *Appro1* is $O(|D|\cdot|q.\psi|+\binom{|q.\psi|}{3}\cdot|D|(\log|D|+l+|q.\psi|)) = O(|D|\cdot|q.\psi|+|q.\psi|^3\cdot|D|(\log|D|+l+|q.\psi|))$. Since l and $|q.\psi|$ are two small numbers in practice, we finally simplify the overall time complexity of *Appro1* as $O(|q.\psi|^3\cdot|D|\log|D|)$.

3.2 Approximate Algorithm 2

Although *Appro1* is a polynomial-time algorithm with a theoretical error bound of a 1.15-factor, when the number of nearby keywords is large, it becomes slow (because its time complexity is *cubic* to the number of nearby keywords). In this section, we propose another approximate algorithm called *Appro2* which is faster than *Appro1*. *Appro2* uses a new concept of "center-centric nearest neighbor set" which is a "looser" concept compared with "minimum covering disk". The reason why we say that "center-centric nearest neighbor set" is looser than "minimum covering disk" is that a set of objects derived from "minimum covering disk" satisfies the concept of "center-centric nearest neighbor set" but the reverse is not true. Since finding "center-centric nearest neighbor set" is computationally cheaper than finding "minimum covering disk" (due to its looser property), *Appro2*

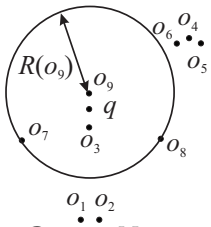


Fig. 4: Center-Centric Nearest Neighbor Set

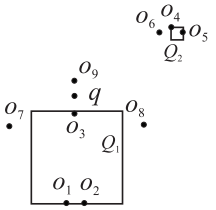


Fig. 5: Minimum ψ-Covering Axis-Parallel Square

(based on "center-centric nearest neighbor set") is faster than *Appro1* (based on "minimum covering disk"). The time complexity of *Appro2* is $O(|q.\psi||D|\log|D|+|D|\log^2|D|)$ which is *linear* to the number of nearby keywords. However, since "center-centric nearest neighbor set" is looser than "minimum covering disk", using "center-centric nearest neighbor set" *alone* in our algorithm may introduce a large error. In our algorithm, in addition to the concept of "center-centric nearest neighbor set", we introduce another new concept called "minimum covering axis-parallel square" which not only is computationally cheap but also minimizes the overall error. Thus, the approximate factor of *Appro2* is just a little bit higher than that of *Appro1*. Specifically, the approximate factor of *Appro2* is 1.79, a little bit greater than the approximate factor of *Appro1* (i.e., 1.15).

Firstly, we give some concepts related to "center-centric nearest neighbor set" and "minimum covering axis-parallel square" before we present our algorithm *Appro2*.

Given an object o in D and a keyword t, we define the *t-keyword center-centric nearest neighbor* of o, denoted by $CCNN(o,t)$, to be the nearest neighbor from o containing keyword t. Given an object o in D, we define the *center-centric nearest neighbor set* of o to be $\cup_{t\in q.\psi}CCNN(o,t)$. Given an object o, we define $R(o)$ to be $\max_{o'\in S}d(o,o')$ where S is the center-centric nearest neighbor set of o. Consider our running example as shown in Figure 2 where $q.t=c$ and $q.\psi=\{a,b\}$. Consider object o_9. The a-keyword center-centric nearest neighbor of o_9 is o_7 and the b-keyword center-centric nearest neighbor of o_9 is o_8. The center-centric nearest neighbor set of o_9 is $\{o_7,o_8\}$. Besides, as shown in Figure 4, $R(o_9)$ is equal to $d(o_9,o_7)$.

We just defined the concept of "center-centric nearest neighbor set". Next, we describe the concept of "minimum covering axis-parallel square".

An *axis-parallel square* is a square such that each side of the square is parallel to one of the axes. Given a set ψ of keywords, a square is said to *cover* ψ if there exists a set of objects in the square covering ψ. Consider our running example. Consider Figure 5. Let $\psi=\{a,b,c\}$. Q_1 is an axis-parallel square covering ψ. Similarly, Q_2 is another axis-parallel square covering ψ. Given a set ψ of keywords, the *minimum ψ-covering axis-parallel square* is defined to be the axis-parallel square such that the square covers ψ and the area of the square is the smallest. Consider Figure 5

again. Q_2 is the minimum ψ-covering axis-parallel square since it covers ψ and its area is the smallest. But, Q_1 is not. Note that the concept of "minimum ψ-covering axis-parallel square" used here is based on the global minimum criterion (meaning that the square has the smallest area globally) but the concept of "minimal covering disk" used in the previous section is based on the local minimal criterion (meaning that the disk has the smallest area locally).

Using Center-Centric Nearest Neighbor Set Only: Next, we present a preliminary version of *Appro2* with the following three steps considering the concept of "center-centric nearest neighbor set" only without the concept of "minimum covering axis-parallel square".

- **Step (a) (Initialization):** A variable o_p, denoting the best-known target object, is initialized as $\emptyset$. A variable O, denoting the best-known nearby set of o_p, is initialized to $\emptyset$. A variable C, denoting the best-known cost, is initialized to ∞.
- **Step (b) (Center-Centric Nearest Neighbor Set Finding):** For each target object o'_p in D, we find the center-centric nearest neighbor set O' of o'_p. Let C' denote $Cost(o'_p|q, O')$. If $C' < C$, then we update O with O', o_p with o'_p, and C with C'.
- **Step (c) (Output):** We return o_p as the output.

Lemma 4. *The* preliminary *version of* Appro2 *is 2-approximate.*

Proof. The proof could be found in [8]. ■

Using Both Center-Centric Nearest Neighbor Set and Minimum Covering Axis-Parallel Square: We would like to lower down the approximate ratio further by involving some additional low-cost operations related to the concept of "minimum covering axis-parallel square". Next, we present the final version of *Appro2*. Before that, we describe the new concept of "critical objects" which is used in the algorithm.

Given an object o in D, o is said to be a *critical object* if o is a target object and $d(q, o) < 0.265 \cdot \frac{1-\alpha}{\alpha} \cdot R(o)$. Roughly speaking, a critical object is "close" to q and how to quantify its closeness is defined in the definition of "critical object". This concrete "closeness" could help us derive a better approximate ratio.

The major reason why the approximate ratio could be lowered down is described as follows. The "best" center-centric nearest neighbor set found by the preliminary version is very near to the optimal solution in some cases but it is a little bit far away from the optimal solution in some other cases. In the final version, we find not only this set but also another set determined based on the concept of "minimum covering axis-parallel square" and the concept of "critical objects" in the final version of *Appro2*. Specifically, the reason why the "best" center-centric nearest neighbor set is a little bit far away from the optimal solution in some cases is that it considers the pairwise distance between a target object and an object from its center-centric nearest neighbor set but does not consider any pairwise distance between any two objects from the center-centric nearest neighbor set, which leads to a larger approximate ratio. Motivated by this weakness, we propose to additionally use the concept of "minimum covering axis-parallel square" and the concept of "critical objects" in the final version of *Appro2* to lower down the approximate ratio. The concept of "minimum covering axis-parallel square" considers the pairwise distance between two objects in a set of objects to be returned (i.e., the second component of Equation (2)), and the concept of "critical objects" considers the distance between q and the target object (i.e., the first component of Equation (2)). Specifically, we efficiently find the minimum $(q.\psi \cup \{q.t\})$-covering axis-parallel square Q in the final version of *Appro2*. Since Q is a square with the smallest area covering both the target keyword and the nearby keywords, using Q could somehow capture the pairwise distance between any two objects in Q, lowering down the second component of Equation (2). Besides, in the final version, we only consider critical objects. Since for each critical object o, $d(q, o) < 0.265 \cdot \frac{1-\alpha}{\alpha} \cdot R(o)$, each object considered by the final version (including object o) is "close" to q, which helps to lower down the first component of Equation (2).

Now, we are ready to present the final version of *Appro2* based on both "center-centric nearest neighbor set" and "minimum covering axis-parallel square". Compared with the preliminary version, this final version considers (1) one additional variable S denoting the set of critical objects, (2) additional operations finding a result based on the concept of "minimum covering axis-parallel square" and (3) an operation of combining the result found based on the concept of "center-centric nearest neighbor set" and the result found based on the concept of "minimum covering axis-parallel square". Details are shown as follows.

- **Step 1 (Initialization):** A variable o_p, denoting the best-known target object, is initialized as $\emptyset$. A variable O, denoting the best-known nearby set of o_p, is initialized to $\emptyset$. A variable C, denoting the best-known cost, is initialized to ∞. A variable S, denoting the set of critical objects, is initialized to $\emptyset$.
- **Step 2 (Center-Centric Nearest Neighbor Set Finding):** For each target object o'_p in D, we perform the following operations. Firstly, we find the center-centric nearest neighbor set O' of o'_p. Let C' denote $Cost(o'_p|q, O')$. If $C' < C$, then we update O with O', o_p with o'_p, and C with C'. Secondly, if o'_p is a critical object, then we insert o'_p into S.
- **Step 3 (Minimum Covering Axis Parallel Square Finding):** Let Θ be $D(q, C)$. We create a new keyword t' and associate t' to each object in S. We find the minimum $(q.\psi \cup \{t'\})$-covering axis-parallel square Q for all objects in Θ by the algorithm proposed in [46]. Let o''_p denote the nearest object in Q containing keyword t' to q. Note that o''_p contains the target keyword (since each object in S contains the target keyword). Let O'' be the nearby set of o''_p in Q.
- **Step 4 (Output):** If $Cost(o''_p|q, O'') < C$, we return o''_p. Otherwise, we return o_p.

Time Complexity: *Appro2* involves four steps, namely Step 1, Step 2, Step 3 and Step 4. Step 1 takes $O(1)$ time. Step 2 takes $O(|D||q.\psi| \log |D|)$ time since there are $O(|D|)$ objects containing the target keyword and each operation of finding the center-centric nearest neighbor set of an object takes $O(|q.\psi| \log |D|)$ time. Step 3 takes $O(|D| \log^2 |D|)$ time since finding the minimum $(q.\psi \cup \{t'\})$-covering axis-parallel square takes $O(|D| \log^2 |D|)$ time [46]. Step 4 takes

Dataset	Web	GN	Hotel
No. of Objects	552,163	1,777,598	20,790
No. of Unique Keywords	2,899,175	222,409	602
No. of Words	249,132,883	18,374,228	80,845

TABLE 3: Dataset Statistics

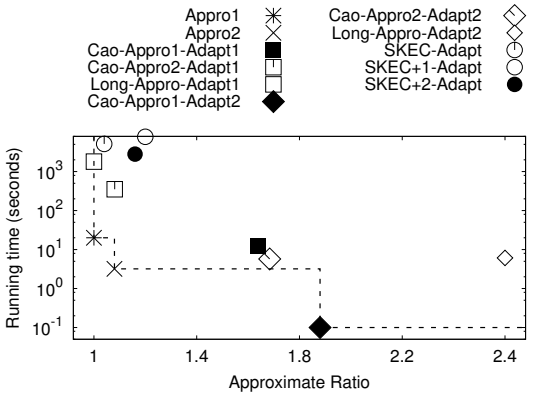


Fig. 6: Trade-off between Accuracy and Efficiency (Web)

$O(1)$ time. So, the total running time is $O(|D||q.\psi| \log |D| + |D| \log^2 |D|)$.

Approximation Ratio: The following shows the approximate factor of *Appro2*.

Theorem 2. Appro2 *returns a 1.79-approximate solution for NSKQ.*

Proof: The proof can be found in [8]

4 EXPERIMENT

4.1 Experimental Set-up

We conducted experiments on a 2.2GHz machine with 4GB RAM installed with a Linux operating system. All algorithms were implemented in C++.

Datasets. We use the same datasets in [5], namely Web, GN and Hotel. Dataset Web was created from two real datasets. The first one, called WEBSPAMUK2007[1], corresponds to a set of web documents. The second one is a set of spatial objects, called TigerCensusBlock[2], 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). Dataset Hotel corresponds to a set of hotels in 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). Table 3 shows the dataset statistics.

Query Generation. For a given dataset D and a positive integer k, we generated queries in the same way as in [5]. We randomly picked a location from the dataspace of D as a query location $q.\lambda$. We sorted all keywords in D in descending order of their frequencies and randomly picked $|q.\psi|$ keywords within the percentile range $[5, 30]$ as $q.\psi$.

1. http://barcelona.research.yahoo.net/webspam/datasets/uk2007
2. http://www.rtreeportal.org

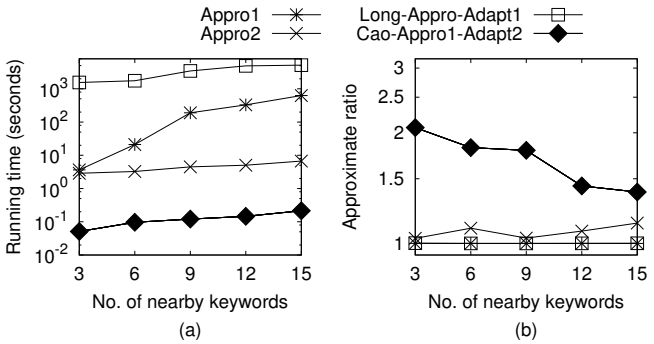


Fig. 7: Effect of $|q.\psi|$ for (Web)

Similarly, we randomly picked a keyword within this range as $q.t$. The default value of $|q.\psi|$ is set to be 6 since users normally do not input more than 6 keywords for query in practice. Besides, the default value of α is set to be 0.5.

Algorithms. We implemented our proposed algorithms, namely *Appro1* and *Appro2*. Besides, we implemented the adapted algorithms as shown in Table 2. We indexed each dataset used with an IR-tree and all algorithms are based on the IR-tree. Since *SKEC+* has a user parameter ϵ with an approximate ratio of $(1.155 + \epsilon)$, we set two values for ϵ here. The first value is 0.01, the default value as suggested in [11]. The second value is a value such that the approximate ratio of *SKEC+* is equal to the approximate ratio of *Appro2* (i.e., 1.79). That is, the second value is set to the difference between 1.79 and 1.155 (i.e., 0.635). Let *SKEC+1-Adapt* and *SKEC+2-Adapt* denote *SKEC+-Adapt* with the first ϵ value setting and the second ϵ value setting, respectively. We also adapted the exact algorithms of CoSKQ to the NSKQ problem and developed our own exact algorithm. But our results showed that they all run more than 1 hour on Web dataset on the default setting. Thus, any exact algorithm fails to fulfill the real-time requirement of NSKQ and ignore the exact algorithms for the clarity in the remaining part.

4.2 Experimental Results

We study the effect of the number of nearby keywords, the average number of keywords per object and the number of objects. Besides, we use two measurements in the experiments: the running time and the approximate ratio. We conducted each experiment with 100 queries and took the average value as the reported value.

To measure the approximate ratio of each approximate algorithm, we adapted existing exact algorithms for CoSKQ to give an exact algorithm for NSKQ. Consider an existing exact algorithm A for CoSKQ. We adapt the existing exact algorithm A in the same way of Adaption 1 of an existing approximate algorithm for CoSKQ described in Section 2.3.

Accuracy vs. Efficiency. Figure 6 shows the trade-off between accuracy and efficiency of all algorithms on the dataset Web where $|q.\psi| = 6$. As shown in the figure, *Appro1*, *Appro2*, *Long-Appro-Adapt1* and *Cao-Appro1-Adapt2* are on the skyline (i.e., any other algorithm must be dominated by one of the above four algorithms.). Thus, in the remaining experiments, we only show the results of *Appro1*, *Appro2*, *Long-Appro-Adapt1* and *Cao-Appro1-Adapt2* for clarity. In Figure 6, the approximate ratio of *Cao-Appro1-Adapt2* is 1.86 but that of *Appro1* or *Appro2* is at most 1.1. As will be shown in the later experiments, the approximate ratio

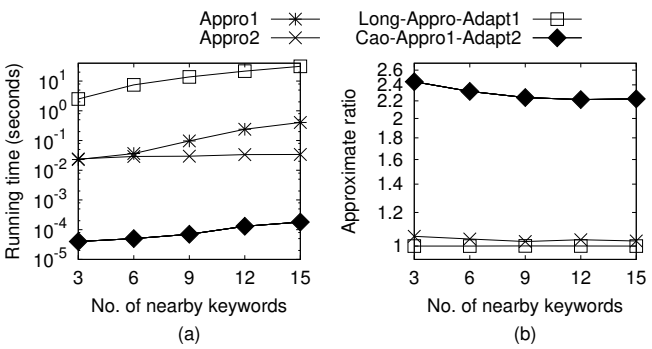


Fig. 8: Effect of $|q.\psi|$ (Hotel)

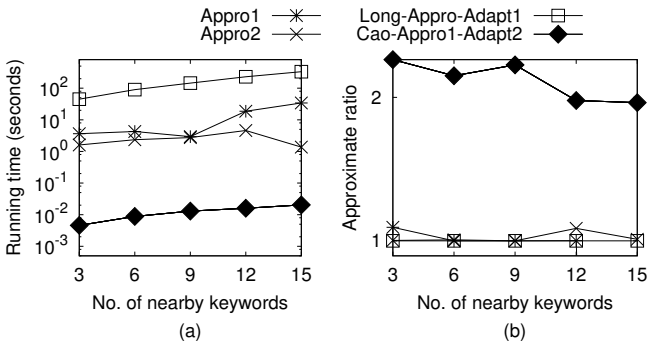


Fig. 9: Effect of $|q.\psi|$ for (GN)

of *Cao-Appro1-Adapt2* is much larger (e.g., 2.8) but that of *Appro1* or *Appro2* is still kept at a very small value (e.g., at most 1.1), which suggests that our proposed algorithms, namely *Appro1* and *Appro2*, are very accurate in practice. Note that there could be a significant difference between a solution with 1.1 approximation ratio and that with 1.86 approximation ratio, though it does not seem to look so, e.g., in the case an optimal solution has its cost of 10km, a 1.1-approximate solution has a cost about 11km and a 1.86-approximate solution about 18.6km, then the difference is about 7.6km (18.6km - 11km) which is 76% of the optimal cost. Figure 6 shows that *Appro1* and *Appro2* ran 2-3 orders of magnitude faster than *Long-Appro-Adapt1*.

Effect of $|q.\psi|$. We tested 5 different values of $|q.\psi|$: 3, 6, 9, 12, 15. The results on the dataset Web are shown in Figure 7. Figures 7(a) and (b) show the running times and the approximate ratios of the approximate algorithms. In Figure 7(a), *Appro1* (resp. *Appro2*) is faster than *Long-Appro-Adapt1* by 1-3 (resp. 2-3) orders of magnitude. In Figure 7(b), the approximate ratio of *Appro1* is nearly 1, which means that its accuracy is significantly high in practice. Consistent with our theoretical analysis, *Appro2* has a smaller approximate ratio than *Cao-Appro1-Adapt2*. The results on Hotel and GN are also similar and are shown in Figure 8 and Figure 9.

Effect of average $|o.\psi|$. We varied the average $|o.\psi|$ on the dataset Hotel, as shown in Figure 10. In dataset Hotel, the average number of keywords per object is nearly 4. We synthesized additional 5 datasets based on this dataset, whose average number of keywords per object are 8, 16, 24, 32, 40. We used the following method to generate the i-th dataset among the additional 5 datasets where $i \in [1,5]$: for every object o in Hotel, we randomly selected another object o' in Hotel and update the keywords of o to $o.\psi \bigcup o'.\psi$. We repeat this procedure $2i$-1 times. It can be verified that this method generates a dataset, whose average $|o.\psi|$ is $8{\cdot}i$. We did the test on the dataset Hotel and the 5 generated datasets.

Figure 10(a) shows that the running time of all algorithms increases with the average $|o.\psi|$. This is because the number $|D'|$ of objects containing at least one keyword in $q.\psi \bigcup \{q.t\}$ increases with the average $|o.\psi|$. Note that since the running time of *Appro1* is cubic to $|q.\psi|$ and linear to $|D'|$ as analyzed, in our experimental result, the running time of *Appro1* is more sensitive to $|q.\psi|$ (as shown in Figure 10(a)) than $|o.\psi|$ (as shown in Figures 7(a), 8(a) and 9(a)). Figure 10(b) shows that the average $|o.\psi|$ has no obvious effect on the approximate ratio of all algorithms. Besides, our proposed algorithms, namely *Appro1* and *Appro2*, are scalable to the average $|o.\psi|$. We also tested the effect of average $|o.\psi|$ on GN and Web and the results are similar to that of Hotel. For the sake of space, we omit their results.

Scalability Test. We tested the scalability of algorithms on 5 synthesized datasets. The number of objects in these datasets are 2M, 4M, 6M, 8M and 10M. We generated the datasets from GN (the largest dataset). We adopted the following method to generate a new dataset DS: we randomly picked an object o in GN. Then, we created a new object o' and assigned a random location to $o'.\lambda$ and assigned $o.\psi$ to $o'.\psi$. Finally, we put o' into DS. We repeated the above procedure until DS had the number of objects as we required.

Figure 11 shows the running times of the algorithms. In the figure, our two approximate algorithms (*Appro1* and *Appro2*) are scalable to the number of objects. It coud be noticed that the runtime is shorter as the number of objects is increased from 2M to 4M (for Appro1) and from 4M to 6M (for Appro2). The reason is stated as follows. In Appro1 and Appro2, we implemented an early stopping strategy to enhance the efficiency which prunes a large number of distant objects and only process objects which are close to the query location. In other words, the running time is not heavily affected by the total number of objects in the dataset but it depends on the spatial distribution of the keywords heavily. The aforementioned issue is due to the spatial keyword distribution more compared with the total number of objects in the dataset. We also conducted the scalability tests on Hotel and Web and their results are similar to that of GN. For the sake of space, we omit their results in the paper.

Effect of α. We tested the effect of α on the Hotel dataset on five different values of α: 0.1, 0.3, 0.5, 0.7, 0.9. The results are as shown in Figure 12. The performances of *Appro1* and *Long-Appro-Adapt1* keep intact under different values of α but the performances of the other two algorithms are affected by α. *Appro1* and *Appro2* are faster than *Long-Appro-Adapt1* by several orders of magnitude. *Appro1* and *Long-Appro-Adapt1* have the best accuracy and their approximate ratio is 1. The approximate ratio of *Appro2* is slightly larger but is still a very small value less than 1.1. *Cao-Appro1-Adapt2* has the worst accuracy and its approximate ratio is up to 1.8.

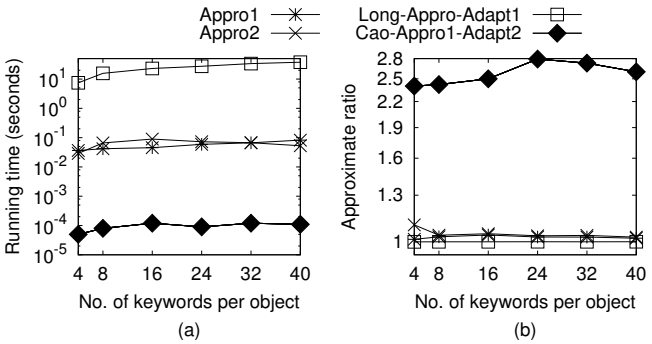


Fig. 10: Effect of average $|o.\psi|$ (Hotel)

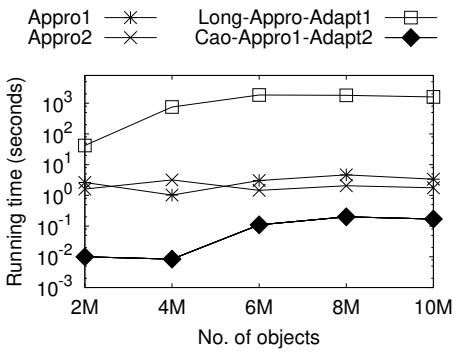


Fig. 11: Scalability Test

Effect of Frequency of Query Keywords. We tested the effect of the frequency of query keywords on the Hotel dataset on five different percentile range: [0-20], [20-40], [40-60], [60-80], [80-100]. Let $\mathcal{O}_i$ denote $\{o \in D | o$ contains a keyword in the ith-Percentile$\}$, where $i \in \{1, 2, 3, 4, 5\}$. The results are as shown in Figure 13. The running time of *Appr1*, *Appro2* or *Long-Appro-Adapt1* keeps almost unaffected under the last 4 percentile ranges but it is larger in the first percentile range. This is because $|\mathcal{O}_1|$ is much larger than $|\mathcal{O}_2|$, $|\mathcal{O}_3|$, $|\mathcal{O}_4|$ and $|\mathcal{O}_5|$, but $|\mathcal{O}_2|$, $|\mathcal{O}_3|$, $|\mathcal{O}_4|$ and $|\mathcal{O}_5|$ are close to each other. The running time of *Cao-Appro1-Adapt2* keeps almost unaffected under all five percentile ranges since its time complexity is logarithmic to the number of objects involved. *Appro1* and *Appro2* are faster than *Long-Appro-Adapt1* by several orders of magnitude. *Appro1* and *Long-Appro-Adapt1* have the best accuracy and their approximate ratios are 1. The approximate ratio of *Appro2* is slightly larger but is still a very small value smaller than 1.1. *Cao-Appro1-Adapt2* has the worst accuracy and its approximate ratio is up to 1.66.

Conclusion. First, *Appro1* and *Appro2* dominate 6 out of 8 existing algorithms in terms of running time and accuracy. Second, *Appro1* and/or *Appro2* are superior over the 2 remaining existing algorithms which are not dominated, namely *Long-Appro-Adapt1* and *Cao-Appro1-Adapt2*. Specifically, *Appro1* and *Appro2* run faster than *Long-Appro-Adapt1* by 1-3 orders of magnitude while providing very similar accurate results and *Appro2* gives more much more accurate results than *Cao-Appro1-Adapt2* and yet run reasonably fast, i.e., within 1 second in most cases. Third, *Appro1* and *Appro2* give slightly different trade-offs between running time and accuracy: *Appro1* does slightly better in accuracy and *Appro2* in running time.

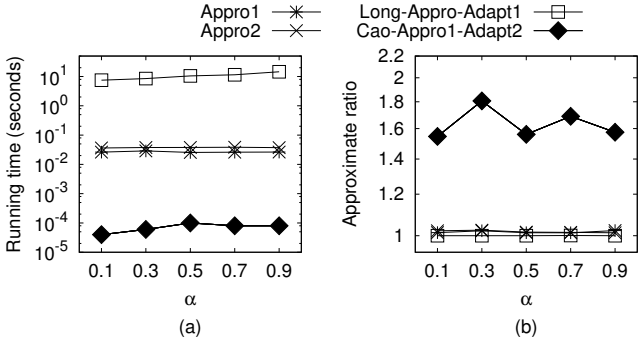


Fig. 12: Effect of α

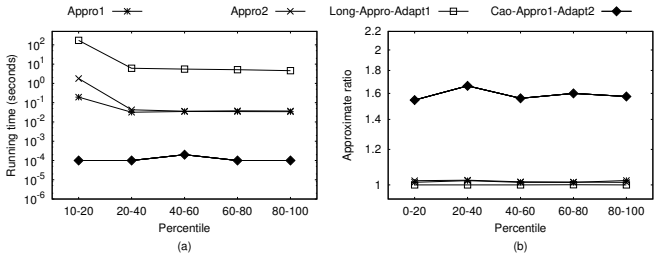


Fig. 13: Effect of Frequency of Query Keywords (Hotel)

5 Conclusion and Future Works

We proposed a new query type called Nearby-fit Spatial Keyword Query (NSKQ) which is shown to be NP-hard. We proposed two approximate algorithms with approximate ratios of 1.115 and 1.79. In experiments, our algorithms significantly outperformed adapted existing methods and were also scalable to large datasets.

There are several potential directions for future work. First, it is interesting to study the continuous NSKQ, in which the query location keeps moving. It is also worth studying the top-k NSKQ Query which returns the best k target objects. Besides, it is an interesting research direction to incorporate the popularity or the rating of each POI into our cost function. Specifically, it will involve the design of a novel cost function which involves objects' locations, keywords and ratings and the design of the corresponding algorithms.

ACKNOWLEDGMENTS: We are grateful to the anonymous reviewers for their constructive comments on this paper. The research of Victor Junqiu Wei and Raymond Chi-Wing Wong is supported by HKRGC GRF 16219816. The research of Cheng Long is partially supported by NTU SUG M4082302.020.

References

[1] I. D. Felipe, V. Hristidis, and N. Rishe, "Keyword search on spatial databases," in *ICDE*, 2008.

[2] A. Cary, O. Wolfson, and N. Rishe, "Efficient and scalable method for processing top-k spatial boolean queries," in *Scientific and Statistical Database Management*. Springer, 2010.

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

[4] T. Lee, J.-w. Park, S. Lee, S.-w. Hwang, S. Elnikety, and Y. He, "Processing and optimizing main memory spatial-keyword queries," *VLDB*, 2015.

[5] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi, "Collective spatial keyword querying," in *SIGMOD*, 2011.

[6] C. Long, R. C.-W. Wong, K. Wang, and A. W.-C. Fu, "Collective spatial keyword queries:a distance owner-driven approach," in *SIGMOD*, 2013.

[7] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi, "Efficient processing of spatial group keyword queries," *TODS*, 2015.

[8] V. J. Wei, R. C.-W. Wong, C. Long, and P. Hui, "On nearby-fit spatial keyword queries (technical report)," in *http://www.cse.ust.hk/~raywong/paper/nearbyFit-technicalReport.pdf*.

[9] D. Zhang, Y. M. Chee, A. Mondal, A. Tung, and M. Kitsuregawa, "Keyword search in spatial databases: Towards searching by document," in *ICDE*, 2009.

[10] D. Zhang, B. C. Ooi, and A. K. H. Tung, "Locating mapped resources in web 2.0," in *ICDE*, 2010.

[11] T. Guo, X. Cao, and G. Cong, "Efficient algorithms for answering the m-closest keywords query," in *SIGMOD*, 2015.

[12] J. Lu, Y. Lu, and G. Cong, "Reverse spatial and textual k nearest neighbor search," in *SIGMOD*, 2011.

[13] F. M. Choudhury, J. S. Culpepper, T. Sellis, and X. Cao, "Maximizing bichromatic reverse spatial and textual k nearest neighbor queries," *VLDB*, 2016.

[14] K. Zheng, H. Su, B. Zheng, S. Shang, J. Xu, J. Liu, and X. Zhou, "Interactive top-k spatial keyword queries," in *ICDE*, 2015.

[15] Y. Lu, J. Lu, G. Cong, W. Wu, and C. Shahabi, "Efficient algorithms and cost models for reverse spatial-keyword k-nearest neighbor search," *TODS*, 2014.

[16] G. Cong, C. S. Jensen, and D. Wu, "Efficient retrieval of the top-k most relevant spatial web objects," *VLDB*, 2009.

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

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

[19] D. Wu, M. L. Yiu, C. S. Jensen, and G. Cong, "Efficient continuously moving top-k spatial keyword query processing," in *ICDE*, 2011.

[20] W. Huang, G. Li, K.-L. Tan, and J. Feng, "Efficient safe-region construction for moving top-k spatial keyword queries," in *CIKM*, 2012.

[21] B. Zheng, K. Zheng, X. Xiao, H. Su, H. Yin, X. Zhou, and G. Li, "Keyword-aware continuous knn query on road networks," in *ICDE*, 2016.

[22] L. Chen, X. Lin, H. Hu, C. S. Jensen, and J. Xu, "Answering why-not questions on spatial keyword top-k queries," in *ICDE*, 2015.

[23] L. Chen, J. Xu, C. S. Jensen, and Y. Li, "Yask: a why-not question answering engine for spatial keyword query services," *VLDB*, 2016.

[24] L. Chen and G. Cong, "Diversity-aware top-k publish/subscribe for text stream," in *SIGMOD*, 2015.

[25] L. Chen, G. Cong, X. Cao, and K.-L. Tan, "Temporal spatial-keyword top-k publish/subscribe," in *ICDE*, 2015.

[26] L. Chen, Y. Cui, G. Cong, and X. Cao, "Sops: A system for efficient processing of spatial-keyword publish/subscribe," *VLDB*, 2014.

[27] L. Chen, G. Cong, and X. Cao, "An efficient query indexing mechanism for filtering geo-textual data," in *SIGMOD*, 2013.

[28] X. Wang, Y. Zhang, W. Zhang, X. Lin, and Z. Huang, "Skype: top-k spatial-keyword publish/subscribe over sliding window," *VLDB*, 2016.

[29] X. Wang, Y. Zhang, W. Zhang, X. Lin, and W. Wang, "Ap-tree: Efficiently support continuous spatial-keyword queries over stream," in *ICDE*, 2015.

[30] ——, "Selectivity estimation on streaming spatio-textual data using local correlations," *VLDB*, 2014.

[31] G. Li, J. Feng, and J. Xu, "Desks: Direction-aware spatial keyword search," in *ICDE*, 2012.

[32] J. Fan, G. Li, L. Zhou, S. Chen, and J. Hu, "Seal: Spatio-textual similarity search," *VLDB*, 2012.

[33] S. Liu, G. Li, and J. Feng, "Star-join: spatio-textual similarity join," in *CIKM*, 2012.

[34] P. Bouros, S. Ge, and N. Mamoulis, "Spatio-textual similarity joins," *VLDB*, 2012.

[35] S. Liu, G. Li, and J. Feng, "A prefix-filter based method for spatio-textual similarity join," *TKDE*, 2014.

[36] K. Deng, X. Li, J. Lu, and X. Zhou, "Best keyword cover search," *TKDE*, 2015.

[37] D.-W. Choi, J. Pei, and X. Lin, "Finding the minimum spatial keyword cover," in *ICDE*, 2016.

[38] J. Liu, K. Deng, H. Sun, Y. Ge, X. Zhou, and C. S. Jensen, "Clue-based spatio-textual query," *VLDB*, 2017.

[39] M. Li, L. Chen, G. Cong, Y. Gu, and G. Yu, "Efficient processing of location-aware group preference queries," in *CIKM*, 2016.

[40] J. B. Rocha-Junior, A. Vlachou, C. Doulkeridis, and K. Nørvåg, "Efficient processing of top-k spatial preference queries," *VLDB*, 2010.

[41] H. K.-H. Chan, C. Long, and R. C.-W. Wong, "Inherent-cost aware collective spatial keyword queries," in *SSTD*, 2017.

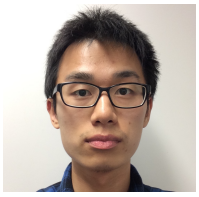
[42] D. Zhang, K.-L. Tan, and A. K. Tung, "Scalable top-k spatial keyword search," in *ICDT*, 2013.

[43] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: an experimental evaluation," in *VLDB*, 2013.

[44] M. De Berg, M. Van Kreveld, M. Overmars, and O. C. Schwarzkopf, *Computational geometry*. Springer, 2000.

[45] P. Afshani and T. M. Chan, "Optimal halfspace range reporting in three dimensions," in *SODA*, 2009.

[46] P. Khanteimouri, A. Mohades, M. A. Abam, and M. R. Kazemi, "Computing the smallest color-spanning axis-parallel square," in *Algorithms and Computation*. Springer, 2013.



Victor Junqiu Wei is currently a researcher in Noah's Ark Lab of Huawei Technologies. He got the PhD degree from the Department of Computer Science and Engineering (CSE) of The Hong Kong University of Science and Technology (HKUST) under supervision of Raymond Chi-Wing Wong. He obtained the BEng in Information Engineering from Nanjing University. His research interest is spatial data management.

Raymond Chi-Wing Wong is an Associate Professor in Computer Science and Engineering (CSE) of The Hong Kong University of Science and Technology (HKUST). He is currently the director of the Risk Management and Business Intelligence (RMBI) program. He was the director of the Computer Engineering (CPEG) program from 2014 to 2016 and was the associate director of the Computer Engineering (CPEG) program from 2012 to 2014. He received the BSc, MPhil and PhD degrees in Computer Science and Engineering in the Chinese University of Hong Kong (CUHK) in 2002, 2004 and 2008, respectively. In 2004-2005, he worked as a research and development assistant under an R&D project funded by ITF and a local industrial company called Lifewood.

Cheng Long is currently an Assistant Professor at the School of Computer Science and Engineering (SCSE), Nanyang Technological University (NTU). From 2016 to 2018, he worked as a lecturer (Asst Professor) at Queen's University Belfast, UK. He got the PhD degree from the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology (HKUST) in 2015. His research interests are broadly in data management, data mining and big data analytics. He has served as a Program Committee member/referee for several top data management and data mining conferences/journals (TODS, VLDBJ, TKDE, ICDM, CIKM, etc.). He is member of IEEE.

Pan Hui is the Nokia Chair in Data Science (with a generous endowment from Nokia) and a Professor of Computer Science at the University of Helsinki. He is also the director of the HKUST-DT System and Media Laboratory (SyMLab) at Computer Science and Engineering Department of Hong Kong University of Science and Technology since January 2013. His current research interests include Mobile Computing, Computer Networking, Data Analytics, and Human-Computer Interactions.

He is also an Associate Editor of both IEEE Transactions on Mobile Computing and IEEE Transactions on Cloud Computing; moreover, he is on the editorial board of the Springer journal of Computational Social Networks, and a guest editor for IEEE Communications Magazine and ACM Transactions on Multimedia Computing, Communications, and Applications. He is an ACM Distinguished Scientist and an IEEE Fellow (Computer Society and Communications Society).