On Generalizing Collective Spatial Keyword Queries

Harry Kai-Ho Chan
The Hong Kong University of Science and Technology
khchanak@cse.ust.hk

Cheng Long
Queen's University Belfast
cheng.long@qub.ac.uk

Raymond Chi-Wing Wong
The Hong Kong University of Science and Technology
raywong@cse.ust.hk

ABSTRACT

With the proliferation of spatial-textual data such as location-based services and geo-tagged websites, spatial keyword queries are ubiquitous in real life. One example of spatial-keyword query is the so-called *collective spatial keyword query* (CoSKQ) which is to find for a given query consisting a query location and several query keywords a set of objects which *covers* the query keywords collectively and has the smallest *cost* wrt the query location. In the literature, many different functions were proposed for defining the *cost* and correspondingly, many different approaches were developed for the CoSKQ problem. In this paper, we study the CoSKQ problem systematically by proposing *a unified cost function* and *a unified approach* for the CoSKQ problem (with the unified cost function). The unified cost function includes all existing cost functions as special cases and the unified approach solves the CoSKQ problem with the unified cost function in a unified way. Experiments were conducted on both real and synthetic datasets which verified our proposed approach.

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 (POI) with textual description (e.g., restaurants, cinema, tourist attractions, and hotels), geo-tagged web objects (e.g., webpages and photos at Flickr), and also geo-social networking data (e.g., users of FourSquare have their check-in histories which are spatial and also profiles which are textual).

One application based on geo-textual data is to search a set of (geo-textual) objects wrt a query consisting of a query location (e.g., the location one is located at) and some textual information (e.g., some keywords expressing the targets one wants to search) such that the objects have their textual information *matching* the query keywords and their locations close to the query location. One scenario of this application is that a tourist wants to find several POIs such that s/he could do sight-seeing, shopping and dining and the POIs are close to his/her location. Another scenario is that a manager wants to set up a project consortium of partners close to each other such that they together offer the capabilities required for successful execution of the whole project.

The above applications were captured by the so-called *Collective Spatial Keyword Query* (CoSKQ) [3, 17, 2] in the literature. Let $\mathcal{O}$ be a set of objects, where 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 $q.\psi$, the CoSKQ problem 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.

In the literature, many different cost functions have been proposed for $cost(S)$ in the CoSKQ problem, and these cost functions are applicable in different scenarios in addition to the above examples. For the CoSKQ problem with each particular cost function, at least one approach has been designed, which we briefly review as follows.

Different cost functions. Five different cost functions have been proposed for the CoSKQ problem, namely, $cost_{Sum}$ [3], $cost_{MaxMax}$ [3], $cost_{MaxMax2}$ [17], $cost_{MinMax}$ [2] and $cost_{SumMax}$ [2]. For example, $cost_{Sum}(S)$ defines the cost to the summation of the distances from the query location to the objects in S, and $cost_{MaxMax}(S)$ defines the cost to a linear combination of the maximum distance between the query location and an object in S and the maximum pairwise distance among the objects in S. The definitions of the rest of cost functions would be introduced later. Each cost function has its own semantic meaning and depending on the application scenario, an appropriate cost function is used.

Different approaches. For the CoSKQ problem with each of these existing cost functions, which was proved to be NP-hard, at least one solution (including an exact algorithm and an approximate algorithm) was developed, and these solutions usually differ from one another. For example, the exact algorithm for the CoSKQ problem with $cost_{Sum}$ is a dynamic programming algorithm [3], while that for the one with $cost_{MaxMax}$ is a branch-and-bound algorithm [3]. Usually, an existing algorithm for the CoSKQ problem with a particular cost function cannot be used to solve that with another cost function.

In this paper, we study the CoSKQ problem systematically by proposing *a unified cost function* and *a unified ap-*

proach for the CoSKQ problem (with the unified cost function).

Without the unified approach, we need to handle different cost functions by different algorithms, which increases the difficulty for CoSKQ to be used in practice. Also, when researchers work on improving the performance of an algorithm, only the corresponding cost function is benefited. Although sometimes it is possible that one algorithm originally designed for one cost function can be adapted for another cost function, the performance of the adapted algorithm is not satisfactory. A better idea is to have a unified cost function and a unified approach, where the unified cost function captures all known cost functions and some other cost functions which are not known before but useful.

Specifically, the main contribution is summarized as follows.

A unified cost function. We propose a unified cost function $cost_{unified}$ which expresses all existing cost functions and a few new cost functions that have not been studied before. The core idea of $cost_{unified}$ is that first two distance components, namely the *query-object distance component* and the *object-object distance component*, are defined, where the former is based on the distances between the query location and those of the objects and the latter is based on the pairwise distances among the set of objects and then $cost_{unified}$ is defined based on the two distance components carefully such that all existing cost functions are captured (Note that this is possible since all ingredients of defining a cost function are distances between the query location and and those distances among objects which are captured by the two components.).

A unified approach. We design a unified approach, which consists of one exact algorithm and one approximate algorithm, for the CoSKQ problem with the unified cost function. For the CoSKQ problem with the cost function instantiated to those existing cost functions, which have been proved to be NP-hard, our exact algorithm is superior over the state-of-the-arts in that it not only has a *unified* procedure, but also runs faster under *all* settings for some cost functions (e.g., $cost_{MinMax}$ and $cost_{MinMax2}$) and under the *majority* of settings for the other cost functions, and our approximate algorithm is always among those algorithms which give the best approximation ratios and runs faster than those algorithms which give similar approximation ratios. For the CoSKQ problem with the cost function instantiated to those new cost functions that have not been studied before, our exact algorithm runs reasonably fast and our approximate algorithm provides certain approximation ratios.

Besides, we conducted extensive experiments based on both real and synthetic datasets which verified our unified approach.

The rest of this paper is organized as follows. Section 2 gives the related work. Section 3 introduces the unified cost function and Section 4 presents the unified approach for CoSKQ. Section 5 gives the empirical study and Section 6 concludes the paper.

2. RELATED WORK

Many existing studies on spatial keyword queries focus on retrieving a *single object* that is close to the query location and relevant to the query keywords.

A *boolean kNN query* [12, 5, 24, 30, 27] finds a list of k objects each covering all specified query keywords. The objects in the list are ranked based on their spatial proximity to the query location.

A *top-k kNN query* [8, 18, 15, 19, 20, 9, 25] adopts the ranking function considering both the spatial proximity and the textual relevance of the objects and returns top-k objects based on the ranking function. This type of queries has been studied on Euclidean space [8, 18, 15], road network databases [19], trajectory databases [20, 9] and moving object databases [25]. Usually, the methods for this kind of queries adopt an index structure called the *IR-tree* [8, 23] capturing both the spatial proximity and the textual information of the objects to speed up the keyword-based nearest neighbor (NN) queries and range queries. In this paper, we also adopt the *IR-tree* for keyword-based NN queries and range queries.

Some other studies on spatial keyword queries focus on finding an *object set* as a solution. Among them, some [3, 17, 2] studied the *collective spatial keyword queries* (CoSKQ). Cao et al. [3, 2] proposed four cost functions, namely $cost_{Sum}$, $cost_{MaxMax}$, $cost_{MinMax}$ and $cost_{SumMax}$, and developed algorithms for the CoSKQ problem with the first three cost functions, leaving that with the fourth cost function, i.e., $cost_{SumMax}$, as future work. Besides, they studied two variations of CoSKQ, namely top-k CoSKQ and weighted CoSKQ, in[2]. Long et al. [17] proposed exact and approximate algorithms for the CoSKQ problem with $cost_{MaxMax}$ and also that with a new cost function $cost_{MaxMax2}$. The details of these cost functions are described in Section 3. In this paper, we also study the CoSKQ problem. Specifically, we propose a *unified* cost function which include all existing cost functions as special cases and based on the unified cost function, we design a *unified* approach, consisting of an exact algorithm and an approximate algorithm.

Another query that is similar to the CoSKQ problem is the *mCK query* [28, 29, 14] which takes a set of m keywords as input and finds m objects with the minimum *diameter* that cover the m keywords specified in the query. In the existing studies of mCK queries, it is usually assumed that each object contains a single keyword. There are some variants of the mCK query, including the *SK-COVER* [7] and the *BKC query* [10]. These queries are similar to the CoSKQ problem in that they also return an object set that covers the query keywords, but they only take a set of keywords as input. In contrast, the CoSKQ problem studied in this paper takes both a set of keywords and a spatial location as inputs.

Skovsgaard et al. [21] proposed a query to find top-k groups of objects with the ranking function considering the spatial proximity and textual relevance of the groups. Liu et al. proposed the *clue-based spatio-textual query* [16] which takes a set of keywords and a clue as inputs, and returns k objects with highest similarities against the clue.

There are also some studies [13, 22] on spatial keyword queries which find an object set in the road network, some [4, 11] which find a *region* as a solution and some [1, 26] which find a *route* as a solution.

3. A UNIFIED COST FUNCTION

Let $\mathcal{O}$ be a set of objects, where 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 two objects o_1 and o_2,

	Parameter			$cost_{unified}(S\|\alpha,\phi_1,\phi_2)$	Existing/New
	$\alpha \in (0,1]$	$\phi_1 \in \{1,\infty,-\infty\}$	$\phi_2 \in \{1,\infty\}$		
a	0.5^*	1	1	$\sum_{o\in S} d(o,q) + \max_{o_1,o_2\in S} d(o_1,o_2)$	$cost_{SumMax}$ [2]
b	0.5^*	1	∞	$\max\{\sum_{o\in S} d(o,q), \max_{o_1,o_2\in S} d(o_1,o_2)\}$	$cost_{SumMax2}$ (New)
c	0.5^*	∞	1	$\max_{o\in S} d(o,q) + \max_{o_1,o_2\in S} d(o_1,o_2)$	$cost_{MaxMax}$ [3, 17, 2]
d	0.5^*	∞	∞	$\max\{\max_{o\in S} d(o,q), \max_{o_1,o_2\in S} d(o_1,o_2)\}$	$cost_{MaxMax2}$ [17]
e	0.5^*	$-\infty$	1	$\min_{o\in S} d(o,q) + \max_{o_1,o_2\in S} d(o_1,o_2)$	$cost_{MinMax}$ [2]
f	0.5^*	$-\infty$	∞	$\max\{\min_{o\in S} d(o,q), \max_{o_1,o_2\in S} d(o_1,o_2)\}$	$cost_{MinMax2}$ (New)
g	1	1	-	$\sum_{o\in S} d(o,q)$	$cost_{Sum}$ [3, 2]
h	1	∞	-	$\max_{o\in S} d(o,q)$	$cost_{Max}$ (New)
i	1	$-\infty$	-	$\min_{o\in S} d(o,q)$	$cost_{Min}$ (New)

* Following the existing studies, $\alpha = 0.5$ is used to illustrate the case of $\alpha \in (0,1)$ for simplicity

Table 1: $cost_{unified}$ under different parameter settings

we denote by $d(o_1,o_2)$ the Euclidean distance between $o_1.\lambda$ and $o_2.\lambda$.

(1) Problem definition. A *collective spatial keyword query* (CoSKQ) [3] is defined as follows.

PROBLEM 1 (COSKQ [3]). *Given a query q with a location $q.\lambda$ and a set of keywords $q.\psi$, **the CoSKQ problem** 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.* □

(2) Existing cost functions. To the best of our knowledge, five cost functions have been proposed for defining $cost(\cdot)$ in the CoSKQ problem, namely $cost_{Sum}$ [3], $cost_{SumMax}$ [2], $cost_{MaxMax}$ [3], $cost_{MaxMax2}$ [17], and $cost_{MinMax}$ [2]. Specifically, these cost functions are defined as follows.

1. $cost_{Sum}$. $cost_{Sum}(S)$ defines the cost to be the summation of the distances from the query location to the objects in S, i.e., $cost_{Sum}(S) = \sum_{o\in S} d(o,q)$.
2. $cost_{SumMax}$. $cost_{SumMax}(S)$ defines the cost to be a linear combination of the summation of distances from the query location to the objects in S and the maximum pairwise distance among the objects in S, i.e., $cost_{SumMax}(S) = \alpha \cdot \sum_{o\in S} d(o,q) + (1-\alpha) \cdot \max_{o_1,o_2\in S} d(o_1,o_2)$, where α represents a real number in $(0,1]$.
3. $cost_{MaxMax}$. $cost_{MaxMax}(S)$ defines the cost to be a linear combination of the maximum distance between the query location and an object in S and the maximum pairwise distance among the objects in S, i.e., $cost_{MaxMax}(S) = \alpha \cdot \max_{o\in S} d(o,q) + (1-\alpha) \cdot \max_{o_1,o_2\in S} d(o_1,o_2)$, where α represents a real number in $(0,1]$.
4. $cost_{MaxMax2}$. $cost_{MaxMax2}(S)$ defines the cost to be the larger one of the maximum distance between the query location and an object in S and the maximum pairwise distance among the objects in S, i.e., $cost_{MaxMax2}(S) = \max\{\max_{o\in S} d(o,q), \max_{o_1,o_2\in S} d(o_1,o_2)\}$.
5. $cost_{MinMax}$. $cost_{MinMax}(S)$ defines the cost to be a linear combination of the minimum distance between the query location and an object in S and the maximum pairwise distance among the objects in S, i.e., $cost_{MinMax}(S) = \alpha \cdot \min_{o\in S} d(o,q) + (1-\alpha) \cdot \max_{o_1,o_2\in S} d(o_1,o_2)$, where α represents a real number in $(0,1]$.

(3) A unified cost function $cost_{unified}$. In this paper, we propose a *unified* cost function $cost_{unified}$ which could be instantiated to many different cost functions including all those five existing ones. Before we give the exact definition of $cost_{unified}$, we first introduce a distance component used for defining $cost_{unified}$, namely the *query-object distance component*. It is defined based on the distances between the query location and the objects in S. Specifically, we denote it by $D_{q,o}(S|\phi_1)$ and define it as follows.

$$D_{q,o}(S|\phi_1) = [\ \sum_{o\in S} (d(o,q))^{\phi_1}\ ]^{\frac{1}{\phi_1}}$$

where $\phi_1 \in \{1,\infty,-\infty\}$ is a user parameter. Depending on the setting of ϕ_1, $D_{q,o}(S|\phi_1)$ corresponds to the summation, the maximum, or the minimum of the distances from the query location to the objects in S. Specifically,

$$D_{q,o}(S|\phi_1) = \begin{cases} \sum_{o\in S} d(o,q), & \text{if } \phi_1 = 1 \\ \max_{o\in S} d(o,q), & \text{if } \phi_1 = \infty \\ \min_{o\in S} d(o,q), & \text{if } \phi_1 = -\infty \end{cases}$$

With the distance component defined, we are ready to introduce the unified cost function $cost_{unified}$. Specifically, we define $cost_{unified}$ as follows.

$$\begin{aligned} &cost_{unified}(S|\alpha,\phi_1,\phi_2) \\ &= \{[\alpha \cdot D_{q,o}(S|\phi_1)]^{\phi_2} + [(1-\alpha) \max_{o_1,o_2\in S} d(o_1,o_2)]^{\phi_2}\}^{\frac{1}{\phi_2}} \end{aligned} \quad (1)$$

where $\alpha \in (0,1]$[1], $\phi_1 \in \{1,\infty,-\infty\}$ and $\phi_2 \in \{1,\infty\}$ are user parameters. In the following, we write $cost_{unified}(S|\alpha,\phi_1,\phi_2)$ simply as $cost(S)$ when there is no ambiguity.

Note that there are many ways to define a unified cost function, while the one we proposed is meaningful and able to cover all existing cost functions.

Under some settings of α, ϕ_1 and ϕ_2, $cost_{unified}$ corresponds to one of the aforementioned existing cost functions (as shown in Table 1). For example, when $\alpha = 1$ and $\phi_1 = 1$ (regardless of the settings of ϕ_2), $cost_{unified}(S)$ corresponds to $cost_{Sum}(S)$ since

$$\begin{aligned} cost_{unified}(S) &= \{[D_{q,o}(S|1)]^{\phi_2}\}^{\frac{1}{\phi_2}} = D_{q,o}(S|1) \\ &= \sum_{o\in S} d(o,q) = cost_{Sum}(S) \end{aligned}$$

and similarly, when $\alpha \in (0,1]$, $\phi_1 = \infty$ and $\phi_2 = 1$, $cost_{unified}(S)$ corresponds to $cost_{MaxMax}(S)$.

[1]In the setting of $\alpha = 0$, the query location has no contribution to the cost. Thus, we do not consider this setting.

Under some other settings of α, ϕ_1 and ϕ_2, $cost_{unified}$ corresponds to a new cost function that has not been studied before. For example, when $\alpha = 0.5$, $\phi_1 = 1$, and $\phi_2 = \infty$, we have

$$cost_{unified}(S) = \{[0.5 \cdot D_{q,o}(S|1)]^{\infty} + [0.5 \cdot \max_{o_1,o_2 \in S} d(o_1, o_2)]^{\infty}\}^{\frac{1}{\infty}}$$
$$= 0.5 \max\{\sum_{o \in S} d(o, q), \max_{o_1,o_2 \in S} d(o_1, o_2)\}$$

where we denote $\max\{\sum_{o \in S} d(o, q), \max_{o_1,o_2 \in S} d(o_1, o_2)\}$ by $cost_{SumMax2}(S)$.

The instantiations of $cost_{unified}$ depending on different parameter settings are shown in Table 1. In the following, we introduce those instantiations that are new.

1. (row b) $cost_{SumMax2}$. The functionality of this cost function is equivalent to the cost function $cost_{Sum}$.

   LEMMA 1. *Let S be an object set. $cost_{SumMax2}(S) = cost_{Sum}(S)$.* □

   PROOF. See [6]. □

2. (row f) $cost_{MinMax2}$. It essentially captures the maximum among two distances, namely the distance between the query location $q.\lambda$ and its nearest object in S and the distance between the two farthest objects in S. A common practice for an individual to explore the objects returned is to visit the object which is the nearest from the query location and explore the others, and thus this cost function is useful when people want to get at their first stop (i.e., the nearest object) fast (this is captured by the query-object distance component) and explore the objects within a small region (this is captured by the farthest pairwise distance of the objects). Compared to the existing cost function $cost_{MinMax}$, $cost_{MinMax2}$ has an advantage that it requires no parameter of α.
3. (row h) $cost_{Max}$. It uses the maximum distance between the query location $q.\lambda$ and an object in S. This cost function is suitable for the scenarios where a user visits one object a time, starting from the query location each time, and wants the worst-case cost as small as possible.
4. (row i) $cost_{Min}$. It uses the distance between the query location $q.\lambda$ and its nearest object in S only, which is of no interest in practice since it put no penalty on those objects that are far away from the query location, e.g., the whole set of objects corresponds to a trivial solution for the CoSKQ problem with $cost_{Min}$. Therefore, we ignore this instantiation of $cost_{unified}$.

(4) Intractability results. It is known that the CoSKQ problem with an existing cost function adopted is NP-hard [3, 17, 2]. That is, the CoSKQ problem is NP-hard under the parameter settings such that $cost_{unified}$ corresponds to an existing cost function. In this paper, we study the intractability of the CoSKQ problem with all possible parameter settings of α, ϕ_1 and ϕ_2 for $cost_{unified}$. Specifically, we have the following result.

THEOREM 1 (INTRACTABILITY). *The CoSKQ problem is NP-hard with all possible parameter settings of α, ϕ_1 and ϕ_2 except for the setting of $\alpha = 1, \phi_1 \in \{\infty, -\infty\}$.* □

PROOF. See [6]. □

4. A UNIFIED APPROACH

In this section, we introduce our unified approach which consists of one exact algorithm called *Unified-E* (Section 4.1) and one approximate algorithm called *Unified-A* (Section 4.2). While the unified cost function combines existing ones, our unified approach is not one which simply combine existing approaches. In fact, both the exact algorithm and approximate algorithm proposed in this paper are clean and elegant while existing approaches have quite different structures.

Before presenting the algorithms, we first give some definitions as follows. Given a query q and an object o in $\mathcal{O}$, we say o is a **relevant object** if $o.\psi \cap q.\psi \neq \emptyset$. We denote $\mathcal{O}_q$ to be the set of all relevant objects. Given a set S of objects, S is said to be a **feasible set** if S covers $q.\psi$ (i.e. $q.\psi \subseteq \cup_{o \in S} o.\psi$). Note that the CoSKQ problem is to find a feasible set with the smallest cost.

Given a non-negative real number r, we denote the circle centered at $q.\lambda$ with radius r by $C(q, r)$. Similarly, the circle centered at $o.\lambda$ with radius r is denoted by $C(o, r)$.

Let q be a query and S be a feasible set. We say that an object $o \in S$ is a **query-object distance contributor** wrt S if $d(o, q)$ *contributes in* $D_{q,o}(S|\phi_1)$. Specifically, in the case of $\phi_1 = 1$ where $D_{q,o}(S|\phi_1) = \sum_{o \in S} d(o, q)$, each object in S is a query-object distance contributor wrt S, in the case of $\phi_1 = \infty$ where $D_{q,o}(S|\phi_1) = \max_{o \in S} d(o, q)$, only those objects in S which have the maximum distance from q are the query-object distance contributors wrt S, and in the case of $\phi_1 = -\infty$ where $D_{q,o}(S|\phi_1) = \min_{o \in S} d(o, q)$, only those objects in S which have the minimum distance from q are the query-object distance contributors wrt S. Then, we define the **key query-object distance contributor** wrt S to the object with the greatest distance from q among all query-object distance contributors wrt S. The concept of "key query-object distance contributor" is inspired by the concept of "query distance owner" proposed in [17], and the concept of "key query-object distance contributor" is more general in the sense that a query distance owner corresponds to a key query distance contributor in the case of $\phi_1 = \infty$ but not in other cases.

Let S be a set of objects and o_i and o_j are two objects in S. We say that o_i and o_j are **object-object distance contributors** wrt S if $d(o_i, o_j)$ *contribute in* $\max_{o,o' \in S} d(o, o')$, i.e, $(o_i, o_j) = \arg\max_{o,o' \in S} d(o, o')$.

Given a query q and a keyword t, the **t-keyword nearest neighbor** of q, denoted by $NN(q, t)$, it defined to be the nearest neighbor (NN) of q containing keyword t. Similarly, $NN(o, t)$ is defined to be the NN of o containing keyword t. Besides, 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., $N(q) = \cup_{t \in q.\psi} NN(q, t)$. Note that $N(q)$ is a feasible set.

4.1 An Exact Algorithm

The idea of *Unified-E* is to iterate through the object-object distance contributors and search for the best feasible set S' in each iteration. This allows CoSKQ with different cost functions to be executed efficiently. Note that each existing algorithm [3, 17, 2] is designed for a specific cost function and they cannot be used to answer CoSKQ with different cost functions. Specifically, *Unified-E* adopts the following search strategy.

- Step 1 (Object-Object Distance Contributors Finding): Select two objects to be the object-object distance contributors wrt the set S' to be constructed;
- Step 2 (Key Query-Object Distance Contributor Finding): Select an object to be the key query-object distance contributor wrt the set S' to be constructed;
- Step 3 (Best Feasible Set Construction): Construct the set S' (which has o_i, o_j as the object-object distance contributors and o_m as the key query-object distance contributor), and update the current best solution $curSet$ with S' if $cost(S') < curCost$, where $curCost$ is the cost of $curSet$;
- Step 4 (Iterative Step): Repeat Step 1 to Step 3 until all possible object-object distance contributors and key query-object distance contributors are iterated.

The above search strategy makes quite effective pruning possible at both Step 1 and Step 2.

Pruning at Step 1. The major idea is that not each relevant objects pair is necessary to be considered as a object-object distance contributor wrt S' to be constructed. First, only the relevant objects in $R_S = C(q, r_1)$ need to be considered, where r_1 is the radius of the region that depends on the parameter setting, as shown in Table 2. It can be proved that if S' contains an object o such that $d(o, q) > r_1$, S' cannot be the optimal solution. Second, we can maintain a lower bound d_{LB} and an upper bound d_{UB} of the distance between the object-object distance contributors for pruning. For example, all those relevant objects pairs (o_i, o_j) with $d(o_i, o_j) > curCost$ (this is because in this case, all those feasible sets S' with (o_i, o_j) as the object-object distance contributor have the cost larger than that of the current best solution, i.e., the best-known cost) could be pruned, i.e., $curCost$ is used as an upper bound. Furthermore, it could be verified easily that when $\phi_1 \in \{1, \infty\}$, all those relevant object pairs (o_i, o_j) with $d(o_i, o_j) < \max_{o \in N(q)} d(o, q) - \min\{d(o_i, q), d(o_j, q)\}$ could be pruned, i.e., $\max_{o \in N(q)} d(o, q) - \min\{d(o_i, q), d(o_j, q)\}$ is used as a lower bound. The details of d_{LB} and d_{UB} for different parameter settings are presented in Table 2. Specifically, we have the following lemma.

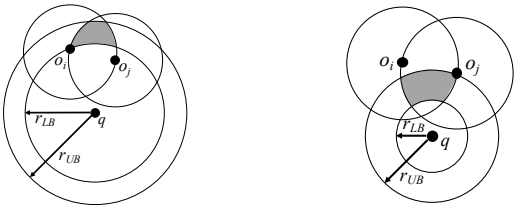
LEMMA 2. *Let o_i and o_j be the object-object distance contributors of the set S to be constructed. For $cost_{unified}$ with different parameter settings, $d(o_i, o_j)$ can be lower bounded by d_{LB} and upper bounded by d_{UB}, as shown in Table 2.* □

PROOF. See [6]. □

Third, given a set having o_i and o_j as the object-object distance contributors, we can compute the lower bound of cost of the set, denoted by $cost(\{o_i, o_j\})_{LB}$, and thus we can prune all those object pairs with $cost(\{o_i, o_j\})_{LB} > curCost$. The details of $cost(\{o_i, o_j\})_{LB}$ for different parameter settings are presented in Table 2. Specifically, we have the following lemma.

LEMMA 3. *Let o_i and o_j be the object-object distance contributors of the set S to be constructed. For $cost_{unified}$ with different parameter settings, $cost(S)$ can be lower bounded by $cost(\{o_i, o_j\})_{LB}$, as shown in Table 2.* □

PROOF. See [6]. □



(a) $\phi_1 \in \{1, \infty\}$ (b) $\phi_1 = -\infty$

Figure 1: Pruning at Step 2 of Unified-E

Pruning at Step 2. The major idea is that not all possible objects in $C(o_i, d(o_i, o_j)) \cap C(o_j, d(o_i, o_j))$ are necessary to be considered for constructing S'. Specifically, we can maintain a lower bound r_{LB} and an upper bound r_{UB} of the distance between the key query-object distance contributors and query. For example, in the case that $\phi_1 = 1$, all those relevant objects o with $d(o, q) < \max\{d(o_i, q), d(o_j, q)\}$ could be safely pruned (this is because such object o can not be the key query-object distance contributor wrt S'), i.e., $\max\{d(o_i, q), d(o_j, q)\}$ is used as lower bound. Figure 1(a) shows the region for the objects to be considered as the key query-object distance contributor. In the case that $\phi_1 = -\infty$, similarly, all those relevant objects o with $d(o, q) > \min\{d(o_i, q), d(o_j, q)\}$ could be safely pruned i.e., $\min\{d(o_i, q), d(o_j, q)\}$ is used as upper bound. Also, all those relevant objects o with $d(o, q) < d_f - d(o_i, o_j)$ could be safely pruned, where $d_f = \max_{o \in N(q)} d(o, q)$ (this is because all those feasible sets S' with o as the key query-object distance contributor have $\max_{o_1, o_2 \in S'} d(o_1, o_2)$ larger than $d(o_i, o_j)$), i.e., $d_f - d(o_i, o_j)$ is used as an lower bound. Figure 1(b) shows the region for the objects to be considered as the key query-object distance contributor. The details of r_{LB} and r_{UB} for different parameter settings are presented in Table 3.

Specifically, we have the following lemma.

LEMMA 4. *Let o_i and o_j be the object-object distance contributors and o_m be the key query-object distance contributors of the set S to be constructed. For $cost_{unified}$ with different parameter settings, $d(o_m, q)$ can be lower bounded by r_{LB} and upper bounded by r_{UB}, as shown in Table 3.* □

PROOF. See [6]. □

With the above search strategy introduced, we present the *Unified-E* algorithm in Algorithm 1. Specifically, we maintain an object set $curSet$ for storing the best-known solution found so far, which is initialized to $N(q)$ (line 1), and $curCost$ to be the cost of $curSet$ (line 2). Recall that $N(q)$ is a feasible set. Then, we initialize R_S to be $C(q, r_1)$ (line 3) and find a set P of all object pairs (o_i, o_j) where o_i and o_j are in R_S to take the roles of object-object distance contributors (line 4).

Second, we perform an iterative process as follows. Consider one iteration. We check whether the lower bound of the set containing o_i and o_j is larger than $curCost$ (line 6). If yes, we stop the iterations (line 7). Otherwise, we proceed to initialize the region R_{ij} to $C(o_i, d(o_i, o_j)) \cap C(o_j, d(o_i, o_j))$ (line 8) and find a set $\mathcal{T}$ of all objects o_m where o_m is in R_{ij} to take the role of key query-object distance contributor (line 9).

Third, we invoke a procedure called *findBestFeasibleSet* (discussed later) for constructing a feasible set S' which takes o_i and o_j as the object-object distance contributors and o_m as the key query-object distance contributor wrt S'

Cost function	Parameter			r_1	d_{LB}	d_{UB}	$cost(\{o_i, o_j\})_{LB}$
	α	ϕ_1	ϕ_2				
$cost_{SumMax}$	0.5	1	1	$curCost$	$d_f - \min\{d(o_i,q), d(o_j,q)\}$	$curCost/2$	$d(o_i,o_j) + d(o_i,q) + d(o_j,q)$
$cost_{MaxMax}$	0.5	∞	1	$curCost$		$curCost - d_f$	$d(o_i,o_j) + \max\{d(o_i,q), d(o_j,q), d_f\}$
$cost_{MaxMax2}$	0.5	∞	∞	$curCost$		$curCost$	$\max\{d(o_i,o_j), d(o_i,q), d(o_j,q), d_f\}$
$cost_{MinMax}$	0.5	$-\infty$	1	$curCost$		$curCost$	$\max\{d(o_i,o_j), d(o_i,q), d(o_j,q), d_f\}$
$cost_{MinMax2}$	0.5	$-\infty$	∞	$2 \cdot curCost$		$curCost$	$\max\{d(o_i,o_j), \max\{d(o_i,q), d(o_j,q)\} - d(o_i,o_j)\}$
$cost_{Sum}$	1	1	-	$curCost$		$curCost$	$d(o_i,q) + d(o_j,q)$

$d_f = \max_{o \in N(q)} d(o,q)$

Table 2: Lower and upper bounds used in Step 1 of Unified-E

Cost function	Parameter			r_{LB}	r_{UB}
	α	ϕ_1	ϕ_2		
$cost_{SumMax}$	0.5	1	1	$\max\{d(o_i,q), d(o_j,q), d_f\}$	$curCost - d(o_i,o_j)$
$cost_{MaxMax}$	0.5	∞	1	$\max\{d(o_i,q), d(o_j,q), d_f\}$	$curCost - d(o_i,o_j)$
$cost_{MaxMax2}$	0.5	∞	∞	$\max\{d(o_i,o_j), d_f\}$	$curCost$
$cost_{MinMax}$	0.5	$-\infty$	1	$d_f - d(o_i,o_j)$	$\min\{curCost - d(o_i,o_j), \min\{d(o_i,q), d(o_j,q)\}\}$
$cost_{MinMax2}$	0.5	$-\infty$	∞	$d_f - d(o_i,o_j)$	$\min\{d(o_i,q), d(o_j,q)\}$
$cost_{Sum}$	1	1	-	$\max\{d(o_i,q), d(o_j,q), d_f\}$	$curCost - d(o_i,q) - d(o_j,q)$

$d_f = \max_{o \in N(q)} d(o,q)$

Table 3: Lower and upper bounds used in Step 2 of Unified-E

Algorithm 1 A Unified Approach (An exact algorithm)

Input: A query q, a set $\mathcal{O}$ of objects and a unified cost function $cost_{unified}(S|\alpha, \phi_1, \phi_2)$

1: $curSet \leftarrow N(q)$
2: $curCost \leftarrow cost(curSet)$
3: $R_S \leftarrow C(q, r_1)$
4: $P \leftarrow$ a set of all relevant object pairs (o_i, o_j) where $o_i, o_j \in R_S$ and $d_{LB} \leq d(o_i, o_j) < d_{UB}$
5: **for** each $(o_i, o_j) \in P$ in ascending order of $cost(\{o_i, o_j\})_{LB}$ **do**
6: **if** $cost(\{o_i, o_j\})_{LB} > curCost$ **then**
7: **break;**
8: $R_{ij} \leftarrow C(o_i, d(o_i, o_j)) \cap C(o_j, d(o_i, o_j))$
9: $\mathcal{T} \leftarrow$ a set of all relevant objects $o_m \in R_{ij}$ where $r_{LB} \leq d(o_m, q) \leq r_{UB}$
10: **for** each $o_m \in \mathcal{T}$ in ascending order of $d(o_m, q)$ **do**
11: $S' \leftarrow$ findBestFeasibleSet(o_i, o_j, o_m)
12: **if** $S' \neq \emptyset$ and $cost(S') < curCost$ **then**
13: $curSet \leftarrow S'$
14: $curCost \leftarrow cost(S')$
15: **return** $curSet$

Algorithm 2 findBestFeasibleSet(o_i, o_j, o_m)

Input: Three objects o_i, o_j, o_m
Output: The feasible set (if any) containing o_i, o_j, o_m with the smallest cost

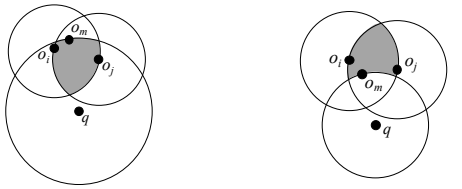
1: $S' \leftarrow \emptyset$
2: $\psi \leftarrow q.\psi - (o_i.\psi \cup o_j.\psi \cup o_m.\psi)$
3: **if** $\psi = \emptyset$ **then**
4: **return** $\{o_i, o_j, o_m\}$
5: **if** $\phi_1 = -\infty$ **then**
6: $R \leftarrow C(o_i, d(o_i, o_j)) \cap C(o_j, d(o_i, o_j)) - C(o_m, d(o_m, q))$
7: **else**
8: $R \leftarrow C(o_i, d(o_i, o_j)) \cap C(o_j, d(o_i, o_j)) \cap C(o_m, d(o_m, q))$
9: $\mathcal{O}' \leftarrow$ a set of all relevant objects in R
10: **if** $\mathcal{O}'$ does not cover ψ **then**
11: **return** $\emptyset$
12: **for** each subset S'' of $\mathcal{O}'$ with $|S''| \leq |\psi|$ **do**
13: **if** S'' covers ψ **then**
14: $S'' \leftarrow S'' \cup \{o_i, o_j, o_m\}$
15: **if** $cost(S'') < cost(S')$ **then**
16: $S' \leftarrow S''$
17: **return** S'

(line 11). Then, we update $curSet$ to S' if S' exists and $cost(S') < curCost$ (lines 12 - 14).

Fourth, we iterate the process with the next relevant object in R_{ij} and with the next object pair from R_S until all relevant objects in R_S have been processed.

Next, we introduce the "findBestFeasibleSet" procedure (used in Algorithm 1), which takes three objects o_i, o_j and o_m as input and finds the best feasible set S' (if any) with the *smallest* cost among all feasible sets which have o_i and o_j as the object-object distance contributors have o_m as a key query-object distance contributor. The procedure is presented in Algorithm 2, and it works as follows. First, it initializes S' as an empty set (line 1). Then, it initializes a variable ψ, denoting the set of keywords in $q.\psi$ not covered by S' yet, as $q.\psi - (o_i.\psi \cup o_j.\psi \cup o_m.\psi)$ (line 2). If $\psi = \emptyset$, then it returns $\{o_i, o_j, o_m\}$ immediately (lines 3-4). Otherwise, it proceeds to retrieve the set $\mathcal{O}'$ containing all relevant objects in R, where R is defined based on the value of ϕ_1 (lines 5-9). When $\phi_1 \in \{1, \infty\}$, $R = C(o_i, d(o_i, o_j)) \cap C(o_j, d(o_i, o_j)) \cap C(o_m, d(o_m, q))$ (line 6), and the region is shown in Figure 2(a). When $\phi_1 = -\infty$, $R = C(o_i, d(o_i, o_j)) \cap C(o_j, d(o_i, o_j)) - C(o_m, d(o_m, q))$ (line 8), and the region is shown in Figure 2(b). The major idea of the region R is that including any object outside the region would violate one or both of the following constraints: (1) o_m is the key query-object distance contributor of the set to be found and (2) o_i and o_j are the object-object distance contributors of the set to be found. If $\mathcal{O}'$ does not cover ψ, it returns $\emptyset$ immediately which implies that no such feasible set could be found (lines 10-11). Otherwise, it finds the target by enumerating all possible subsets S'' of $\mathcal{O}'$ with size at most $|\psi|$ (by utilizing the inverted lists maintained for each keyword in ψ), and for each possible S'', if it covers ψ and $cost(S'' \cup \{o_i, o_j, o_m\}) < cost(S')$, S' is updated correspondingly (lines 12-16).

We also develop some other pruning techniques based on a concept of "dominance" for further improving the efficiency of the algorithm. The major idea is that under some parameter settings, the solution of the CoSKQ problem contains only those objects that are not dominated by other objects.



(a) $\phi_1 \in \{1, \infty\}$ (b) $\phi_1 = -\infty$

Figure 2: Search space R in Algorithm 2

Details could be found in [6].

Time complexity analysis. Let $|P|$ be the number of object pairs in P. Note that $|P|$ is usually much smaller than $|\mathcal{O}_q|^2$ since $|P|$ corresponds to the number of relevant objects we process in R_S and the area occupied by R_S is typically small. Let $|R_{ij}|$ be the number of relevant objects in R_{ij}. The time complexity of Algorithm 1 is $O(|P|\cdot|R_{ij}|\cdot\theta)$, where θ is the time complexity of Algorithm 2. It could be verified that θ is dominated by the step of enumerating the object sets (lines 12-16 in Algorithm 2), whose cost is $O(|\mathcal{O}'|^{|q.\psi|-3} \cdot |\psi|^2)$ since it searches at most $O(|\mathcal{O}'|^{|q.\psi|-3})$ subsets S'' that cover ψ and the checking cost for each subset is $O(|\psi|^2)$. As a result, the time complexity of *Unified-E* is $O(|P| \cdot |R_{ij}| \cdot |\mathcal{O}'|^{|q.\psi|-3} \cdot |\psi|^2)$.

4.2 An Approximate Algorithm

In this part, we introduce the approximate algorithm *Unified-A*. Compared with *Unified-E*, *Unified-A* drops the step of object-object distance contributors finding and replaces the step of best feasible set construction which is expensive with a step of (arbitrary) feasible set construction which is efficient, and thus it enjoys significantly better efficiency. Specifically, the *Unified-A* adopts the following search strategy.

- Step 1 (Key Query-Object Distance Contributor Finding): Select a relevant object o to be key query-object distance contributor wrt a set S' to be constructed;
- Step 2 (Feasible Set Construction): Construct the set S' (which has o as a key query-object distance contributor);
- Step 3 (Optimal Set Updating): Update the current best solution *curSet* if $cost(S') < curCost$, where *curCost* is the cost of *curSet*;
- Step 4 (Iterative Step): Repeat Step 1 to Step 3 until all possible key query-object distance contributors are iterated.

The above search strategy makes quite effective pruning possible at both Step 1 and Step 2.

Pruning at Step 1. The major idea is that not each relevant object is necessary to be considered as a key query-object distance contributor wrt S' to be constructed. Specifically, in the case of $\phi_1 \in \{1, \infty\}$, all those relevant objects o with $d(o, q) > curCost$ (this is because all those feasible sets S' with o as a key query-object distance contributor have the cost larger than the best-known cost *curCost*, and thus they could be pruned) or $d(o, q) < \max_{o \in N(q)} d(o, q)$ (this is because there exist no feasible sets within the disk of $C(q, \max_{o \in N(q)} d(o, q) - \epsilon)$ where ϵ is close to zero) could be pruned. Therefore, we can maintain a region R which corresponds to the "ring region" enclosed by $C(q, curCost)$ and $C(q, \max_{o \in N}(q) d(o, q))$ for pruning the search space at Step 1. In the case of $\phi_1 = -\infty$, the region R could also be defined correspondingly. Details of the region R for different parameter settings are presented in Table 4.

Algorithm 3 A Unified Approach (An approximate algorithm)

Input: A query q, a set $\mathcal{O}$ of objects and a unified cost function $cost_{unified}(S|\alpha, \phi_1, \phi_2)$

```
1: curSet ← N(q)
2: curCost ← cost(curSet)
3: Initialize the region R
4: for each relevant object o ∈ R in ascending order of d(o, q) do
5:    Initialize the region R_o
6:    S' ← findFeasibleSet(o, R_o)
7:    if S' ≠ ∅ and cost(S') < curCost then
8:       curSet ← S'
9:       curCost ← cost(S')
10: return curSet
```

Pruning at Step 2. The major idea is that not all possible objects in R_o are necessary to be considered for constructing S'. Specifically, in the case of $\phi_1 \in \{1, \infty\}$, all those relevant objects outside $C(q, d(o, q))$ could be safely pruned (this is because including one such object would fail o to be a key query-object distance contributor wrt S'). Thus, we can maintain a region R_o which corresponds to $C(q, d(o, q))$ for pruning the search space at Step 2. In the case of $\phi_1 = -\infty$, the region R_o could also be maintained appropriately. Details of the region R_o for different parameter settings are presented in Table 4 as well.

With the above search strategy and pruning techniques introduced, the *Unified-A* algorithm is presented in Algorithm 3. Specifically, we maintain an object set *curSet* for storing the best-known solution found so far, which is initialized to $N(q)$ (line 1) and *curCost* to be the cost of *curSet* (line 2). Then, we perform an iterative process for each relevant object $o \in R$ in ascending order of $d(o, q)$ (lines 3-4). Consider one iteration. First, we initialize the region R_o (line 5). Second, we invoke a procedure called *findFeasibleSet* (discussed later) for constructing a feasible set S' which takes o as a key query-object distance contributor wrt S' (line 6). Third, we update *curSet* to S' and *curCost* to $cost(S')$ if S' exists and $cost(S') < curCost$ (lines 7-9). We iterate the process with the next relevant object from R which has not been processed until all relevant objects in R have been processed.

Next, we introduce the "findFeasibleSet" procedure (used in Algorithm 3), which takes an object o and a region R_o as input and finds a feasible set S' (if any) which contains objects in R_o (including o) and has o as a key query-object distance contributor. The procedure is presented in in Algorithm 4, and it is similar to the "findBestFeasibleSet" procedure (in Algorithm 2) except that it replaces the enumeration process with an iterative process (lines 8-14) for searching for a feasible set.

Depending on the value of ϕ_1, the algorithm uses different criterion for picking an object at an iteration, which is described as follows.

Case 1: $\phi_1 = 1$. It picks the object which has the smallest ratio of its distance to q to the number of remaining keywords covered. Using this criterion, the algorithm tries to

Cost function	Parameter			R	R_o
	α	ϕ_1	ϕ_2		
$cost_{SumMax}$	0.5	1	1	$C(q, curCost) - C(q, d_f)$	$C(q, d(o, q))$
$cost_{MaxMax}$	0.5	∞	1	$C(q, curCost) - C(q, d_f)$	$C(q, d(o, q))$
$cost_{MaxMax2}$	0.5	∞	∞	$C(q, curCost) - C(q, d_f)$	$C(q, d(o, q))$
$cost_{MinMax}$	0.5	$-\infty$	1	$C(q, curCost)$	$C(q, curCost) \cap C(o, curCost - d(o, q)) - C(q, d(o, q))$
$cost_{MinMax2}$	0.5	$-\infty$	∞	$C(q, 2 \cdot curCost)$	$C(q, 2 \cdot curCost) \cap C(o, curCost - d(o, q)) - C(q, d(o, q))$
$cost_{Sum}$	1	1	-	$C(q, curCost) - C(q, d_f)$	$C(q, d(o, q))$

$d_f = \max_{o \in N(q)} d(o, q)$

Table 4: R and R_o in Unified-A

Algorithm 4 findFeasibleSet(o, R_o)

Input: An object o, a region R_o
Output: A feasible set (if any) containing objects in R_o (including o)

$S' \leftarrow \{o\}$
$\psi \leftarrow q.\psi - o.\psi$
if $\psi = \emptyset$ **then**
  return S'
$\mathcal{O}' \leftarrow$ a set of all relevant objects in R_o
if $\mathcal{O}'$ does not cover ψ **then**
  return $\emptyset$
while $\psi \neq \emptyset$ **do**
  if $\phi_1 = 1$ **then**
    $o' \leftarrow \arg\min_{o' \in \mathcal{O}'} \frac{d(o', q)}{|\psi \cap o'.\psi|}$
  else
    $o' \leftarrow \arg\min_{o' \in \mathcal{O}'} d(o', o)$ and $\psi \cap o'.\psi \neq \emptyset$
  $S' \leftarrow S' \cup \{o'\}$
  $\psi \leftarrow \psi - o'.\psi$
return S'

pick objects in a way that minimizes the sum of the distances between the query location and the objects.

Case 2: $\phi_1 \in \{\infty, -\infty\}$. It picks the object which is the nearest to o and covers some of the uncovered keywords. Using this criterion, the algorithm tries to pick objects in a way that minimizes the maximum pairwise distance between the objects.

We also develop two techniques based on the concept of *information re-use* for implementing the *Unified-A* with better efficiency. The details could be found in [6].

Time complexity analysis. Let $|R|$ be the number of relevant objects in R. It could be verified that the complexity of the "findFeasibleSet" (Algorithm 4) is $O(|\psi| \cdot |\mathcal{O}'| \log |\mathcal{O}'|)$ (note that a heap structure with $|\mathcal{O}'|$ elements could be used and there are at most $O(|\psi|)$ operations based on the heap). Therefore, the time complexity of *Unified-A* is $O(|R| \cdot |\psi| \cdot |\mathcal{O}'| \log |\mathcal{O}'|)$.

Approximation ratio analysis. In general, the *Unified-A* algorithm gives different approximation ratios for different parameter settings, which are given in the following theorem.

THEOREM 2. *The Unified-A algorithm gives approximation ratios as shown in Table 5 for the CoSKQ problem under different parameter settings.* □

PROOF. See [6]. □

According to the results in Table 5, we know that in despite of the fact that our unified approach is designed for a unified cost function which could be instantiated to many different cost functions, the approximate algorithm based on the unified approach provides *better* (same) approximation ratios than (as) the state-of-the arts for three (two) existing cost functions.

Cost function	Parameter			Unified-A	Best known
	α	ϕ_1	ϕ_2	Appro. ratio	Appro. ratio
$cost_{MinMax}$	0.5	$-\infty$	1	2	3 [2]
$cost_{MinMax2}$	0.5	$-\infty$	∞	2	N.A.
$cost_{Sum}$	1	1	-	$H_{\|\psi\|}$	$H_{\|q.\psi\|}$ [2]
$cost_{SumMax}$	0.5	1	1	$2H_{\|q.\psi\|}$	N.A.
$cost_{SumMax2}$	0.5	1	∞	$H_{\|\psi\|}$	$H_{\|q.\psi\|}$ [2]
$cost_{MaxMax}$	0.5	∞	1	1.375	1.375 [17]
$cost_{MaxMax2}$	0.5	∞	∞	$\sqrt{3}$	$\sqrt{3}$ [17]
$cost_{Max}$	1	∞	-	1	N.A.
$cost_{Min}$	1	$-\infty$	-	1	N.A.

Table 5: Approximation ratios of Unified-A and existing solutions

	Hotel	GN	Web
Number of objects	20,790	1,868,821	579,727
Number of unique words	602	222,409	2,899,175
Number of words	80,645	18,374,228	249,132,883

Table 6: Datasets used in the experiments

Cost function	Exact Algorithm	Appro. Algorithm
$cost_{MinMax}$	Cao-E1 [2]	Cao-A1 [2]
$cost_{MinMax2}$	Cao-E1 [2]*	Cao-A1 [2]*
$cost_{Sum}$	Cao-E2 [2]	Cao-A3 [2]
$cost_{SumMax}$	Cao-E1 [2]*	Cao-A3 [2]*
$cost_{MaxMax}$	Cao-E1 [2], Long-E [17]	Cao-A1 [2], Cao-A2 [2], Long-A [17]
$cost_{MaxMax2}$	Cao-E1 [2]*, Long-E [17]	Cao-A1 [2]*, Cao-A2 [2]*, Long-A [17]

Table 7: Algorithms for comparison (those with the asterisk symbol are adaptations)

5. EMPIRICAL STUDIES

5.1 Experimental Set-up

Datasets. Following the existing studies [3, 17, 2], we used three real datasets in our experiments, namely Hotel, GN and Web. Dataset Hotel contains a set of hotels in the U.S. (www.allstays.com), each of which has a spatial location and a set of words that describe the hotel (e.g., restaurant, pool). Dataset GN was collected from the U.S. Board on Geographic Names (geonames.usgs.gov), where each object has a location and also a set of descriptive keywords (e.g., a geographic name such as valley). Dataset Web was generated by merging two real datasets. One is a spatial dataset called TigerCensusBlock[2], which contains a set of census blocks in Iowa, Kansas, Missouri and Nebraska. The other is WEBSPAM-UK2007[3], which consists of a set of web documents. Table 6 shows the statistics of the three datasets.

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

Query Generation. Let O be a dataset of objects. Given an integer k, we generate a query q with k query keywords similarly as [3, 17] did. Specifically, to generate $q.\lambda$, we randomly pick a location from the MBR of the objects in O, and to generate $q.\psi$, we first rank all the keywords that are associated with objects in O in descending order of their frequencies and then randomly pick k keywords in the percentile range of [10, 40].

Cost functions. We study all instantiations of our unified cost function except for $cost_{Min}$ and $cost_{SumMax2}$ since as we mentioned in Section 3, the former is of no interest and the latter is equivalent to $cost_{Sum}$. That is, we study 7 cost functions in total, namely $cost_{MinMax}$, $cost_{MinMax2}$, $cost_{Sum}$ and $cost_{SumMax2}$, $cost_{MaxMax}$, $cost_{MaxMax2}$ and $cost_{Max}$.

Algorithms. Both the *Unified-E* algorithm and the *Unified-A* algorithm are studied. For comparison, for the CoSKQ problem with an existing cost function, the state-of-the-art algorithms are used and for the CoSKQ problem with a new cost function, some adaptions of existing algorithms are used. The state-of-the-art algorithms are presented in Table 7, where *Cao-E1*, *Cao-E2*, *Cao-A1*, *Cao-A2* and *Cao-A3* refer to the algorithms *MAXMAX-Exact*, *SUM-Exact*, *MAXMAX-Appro1*, *MAXMAX-Appro2* and *SUM-Appro* [2], respectively, and *Long-E* and *Long-A* refer to the algorithms *MaxSum-Exact* and *MaxSum-Appro* [17], respectively. Note that though the cost function $cost_{SumMax}$ was proposed in [2], it was left as future work to develop solutions and thus we adapt some existing algorithms for the CoSKQ problem with this cost function.

All experiments were conducted on a Linux platform with a 2.66GHz machine and 32GB RAM. The IR-tree index structure is memory resident.

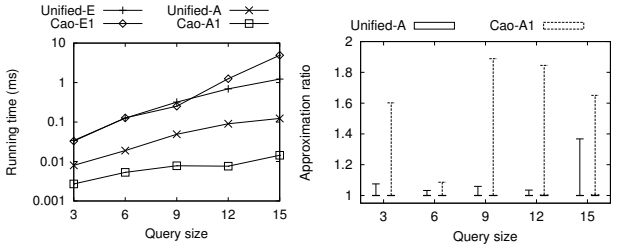
5.2 Experimental Results

Following the existing studies [3, 17, 2], we used the running time and the approximation ratio (for approximate algorithms only) as measurements. Note that different sets of objects with the same costs are treated equally, and thus precision or recall are not used as measures in our experiments. For each experimental setting, we generated 50 queries, and ran the algorithms with each of these 50 queries. The average running times are reported, while box plot is used to show the approximation ratios.

5.2.1 Effect of $|q.\psi|$

Following the existing studies [3, 17], we vary the number of query keywords (i.e., $|q.\psi|$) from $\{3, 6, 9, 12, 15\}$. The results on the dataset Hotel are presented and those on the datasets GN and Web are similar and could be found in the full version of the paper [6].

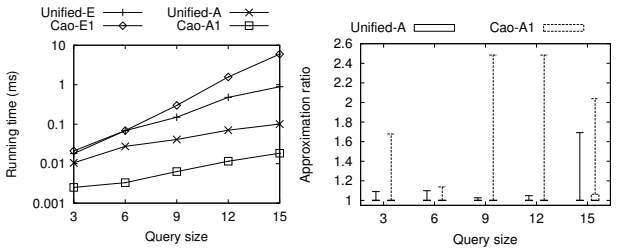
(1) $cost_{MinMax}$**.** The results for $cost_{MinMax}$ are shown in Figure 3. According to Figure 3(a), the running time of each algorithm increases when $|q.\psi|$ increases. Our exact algorithm *Unified-E* runs consistently faster than the state-of-the-art algorithm *Cao-E1* and the gap becomes larger when $|q.\psi|$ increases. This could be explained by the fact *Cao-E1* performs the expensive exhaustive search on the pivot objects whose number increases fast with $|q.\psi|$ while *Unified-E* only need to search on the regions that are possible to contain the object sets. Besides, our approximate algorithm *Unified-A* runs quite fast, e.g., less than 0.1 seconds, though it is slower than *Cao-A1*. According to Figure 3(b), *Unified-A* has its approximation ratios consistently better than *Cao-A1*, e.g., for 4 out of 5 settings (3, 6, 9 and 12), the largest approximation ratios of *Unified-A* is *at most* 1.075 while for other 4 out of 5 settings (3, 9, 12 and 15), the largest approximation ratios of *Cao-A1* is *at least* 1.602 (and up to 1.889). Note that there could be an significant difference between a solution with 1.075 approximation ratio and that with 1.602 approximation ratio, though it does not seem to look so, e.g., in the case an optimal solution has its cost of 10km, a 1.075-approximate solution has a cost about 11km and a 1.602-approximate solution about 16km, then the difference is about 5km (16km - 11km) which is almost half of the optimal cost. The reason could be that *Unified-A* performs an iterative process on the key query-object distance contributor which helps improve the approximation ratio while *Cao-A1* does not. Besides, we note that the approximation ratio of *Unified-A* is exactly 1 for more than 90% queries, while that of *Cao-A1* is only 60%.



(a) Running time (b) Approximation ratio

Figure 3: Effect of $|q.\psi|$ on $cost_{MinMax}$ (Hotel)

(2) $cost_{MinMax2}$**.** The results for $cost_{MinMax2}$ are shown in Figure 4, which are similar to those for $cost_{MinMax}$, i.e., *Unified-E* runs consistently faster than *Cao-E1* and *Unified-A* gives better approximation ratios than *Cao-A1* with reasonable efficiency.



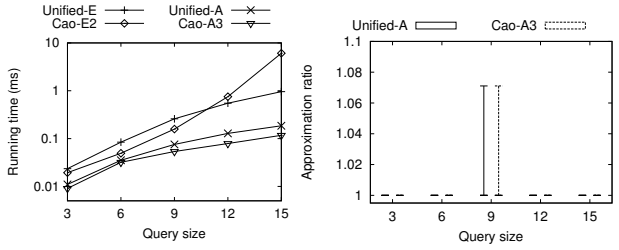
(a) Running time (b) Approximation ratio

Figure 4: Effect of $|q.\psi|$ on $cost_{MinMax2}$ (Hotel)

(3) $cost_{Sum}$**.** The results for $cost_{Sum}$ are shown in Figure 5. According to Figure 5(a), *Unified-E* runs similarly fast as *Cao-E2* when $|q.\psi| \leq 9$ and runs faster than *Cao-E2* when $|q.\psi| > 9$. *Unified-E* has a very restrict search space, e.g., only those dominant objects, and *Cao-E2* is a dynamic programming algorithm which might be more sensitive to $|q.\psi|$. Besides, *Unified-A* has a very similar running time as *Cao-A3*. According to Figure 5(b), *Unified-A* and *Cao-A3* give very similar approximation ratios.

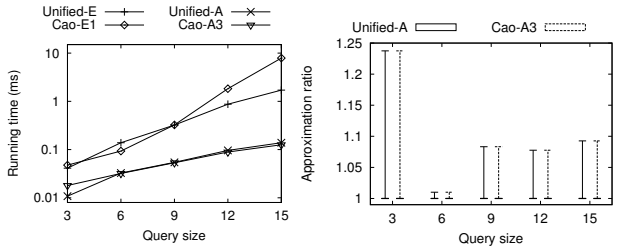
(4) $cost_{SumMax}$**.** The results for $cost_{SumMax}$ are shown in Figure 6, which are similar to those for $cost_{Sum}$ except that the competitor is *Cao-E1*, i.e., *Unified-E* runs faster than *Cao-E1* when $|q.\psi|$ grows and *Unified-A* has similar running time and also approximation ratios as *Cao-A3*.

(5) $cost_{MaxMax}$**.** The results for $cost_{MaxMax}$ are shown in Figure 7. According to Figure 7(a), each algorithm has its



(a) Running time (b) Approximation ratio

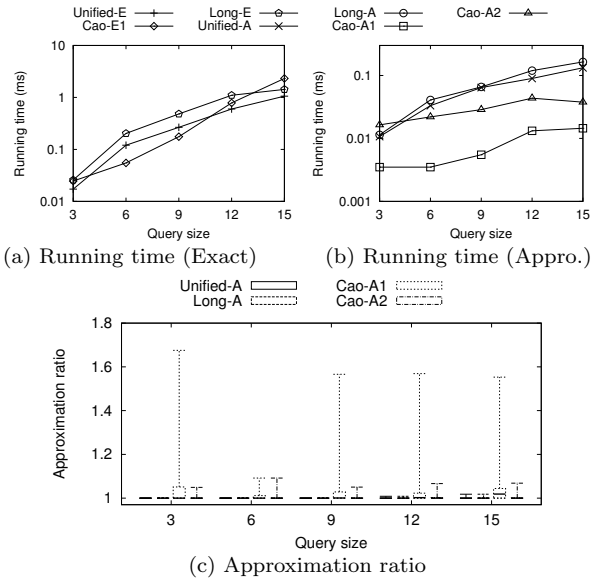
Figure 5: Effect of $|q.\psi|$ on $cost_{Sum}$ (Hotel)



(a) Running time (b) Approximation ratio

Figure 6: Effect of $|q.\psi|$ on $cost_{SumMax}$ (Hotel)

running time grows when $|q.\psi|$ increases (in particular, *Cao-E1* has its running time grows the fastest). Besides, *Unified-E* runs consistently faster than *Long-E* and runs faster than *Cao-E1* as well when $|q.\psi|$ gets larger. According to Figure 7(b), all approximate algorithms including *Unified-A* run fast, e.g., less then 0.1 seconds, and according to Figure 7(c), *Unified-A* is one of two algorithms that give the best approximation ratio (the other is *Long-A*). Note that *Unified-A* runs consistently faster than *Long-A*, and the reason could be that *Unified-A* has computation strategies based on information re-use while *Long-A* does not. The largest approximation ratios of *Unified-A* is only 1.018, while that of *Cao-A1* and *Cao-A2* could be up to 1.686 and 1.092, respectively. Besides, *Unified-A* gives approximation ratio of exactly 1 for 97% queries, while that of *Cao-A1* and *Cao-A2* are 44% and 76%, respectively.

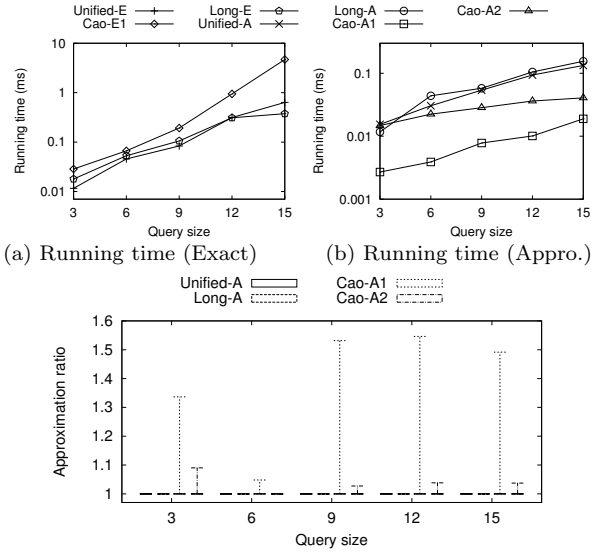


(a) Running time (Exact) (b) Running time (Appro.)

(c) Approximation ratio

Figure 7: Effect of $|q.\psi|$ on $cost_{MaxMax}$ (Hotel)

(6) $cost_{MaxMax2}$**.** The results for $cost_{MaxMax2}$ are shown in Figure 8, which are similar as those for $cost_{MaxMax}$, i.e., *Unified-E* has the best efficiency in general and *Unified-A* is among one of the two algorithms which give the best approximation ratios and also run reasonably fast. Note that *Unified-A* is able to return an optimal solution in all queries, while the largest approximation ratios of *Cao-A1* and *Cao-A2* are 1.546 and 1.090, respectively.



(a) Running time (Exact) (b) Running time (Appro.)

(c) Approximation ratio

Figure 8: Effect of $|q.\psi|$ on $cost_{MaxMax2}$ (Hotel)

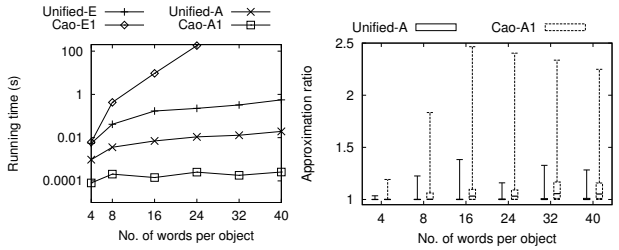
(7) $cost_{Max}$**.** The results for $cost_{Max}$ are shown in Figure 15(a). According to the results, both *Unified-E* and *Unified-A* run very fast, e.g., they ran less than 0.01 ms for all settings of $|q.\psi|$. This is mainly because that both algorithms essentially find $N(q)$ as the solution.

5.2.2 *Effect of average* $|o.\psi|$

We further generated 5 datasets based on the Hotel dataset, where the average number of keywords an object contains (i.e. average $|o.\psi|$) is close to 8, 16, 24, 32, and 40, respectively. In the Hotel dataset, the average number of keywords an object contains is close to 4. To generate a dataset with its average $|o.\psi|$ equal to 8, we do the following. For each object o in the Hotel dataset, we augment $o.\psi$ by including all those keywords in $o'.\psi$ to $o.\psi$ (i.e., $o.\psi \leftarrow o.\psi \cup o'.\psi$) where o' is a randomly picked object. To generate the datasets with the average $|o.\psi|$ equal to 16, 24, 32 and 40, we repeat the above process appropriate times. We vary average $|o.\psi|$ from $\{4, 8, 16, 24, 32, 40\}$ and following [2], we use the default setting of $|q.\psi| = 10$.

(1) $cost_{MinMax}$**.** The results for $cost_{MinMax}$ are shown in Figure 9, where the results of running time of *Cao-E1* for $|o.\psi| \geq 32$ are not shown simply because it ran for more than 10 hours (this applies for all the following results). According to Figure 9(a), all algorithms except for *Cao-E1* are quite scalable when $|o.\psi|$ grows. The poor scalability of *Cao-E1* could be due to the fact that *Cao-E1* is based on the search space of relevant objects around the candidate objects, which grows rapidly when $|o.\psi|$ increases. Besides, our exact algorithm *Unified-E* runs consistently better than *Cao-E1* and *Unified-A* runs fast, though not as fast as *Cao-A1*, and gives obviously better approximation ratios than *Cao-A1* (Figure 9(b)). Specifically, the largest approximation ratios of *Unified-A* is only 1.383, which is small, while

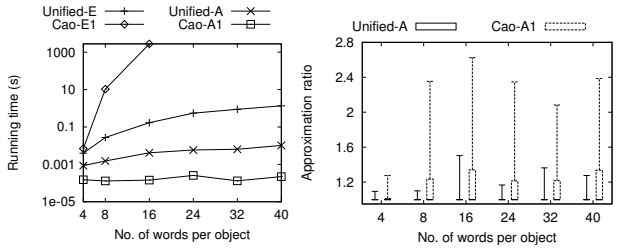
that of *Cao-A1* is up to 2.465, which is not suitable for practical use.



(a) Running time (b) Approximation ratio

Figure 9: Effect of average $|o.\psi|$ on $cost_{MinMax}$

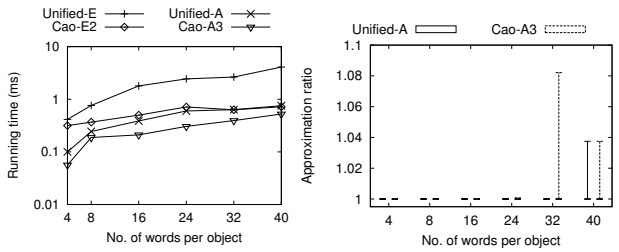
(2) $cost_{MinMax2}$**.** The results for $cost_{MinMax2}$ are shown in Figure 10, which are similar to those for $cost_{MinMax}$, i.e., all algorithms except for *Cao-E1* are scalable when $|o.\psi|$ grows, *Unified-E* runs consistently faster than *Cao-E1*, and *Unified-A* runs fast and gives the best approximation ratios.



(a) Running time (b) Approximation ratio

Figure 10: Effect of average $|o.\psi|$ on $cost_{MinMax2}$

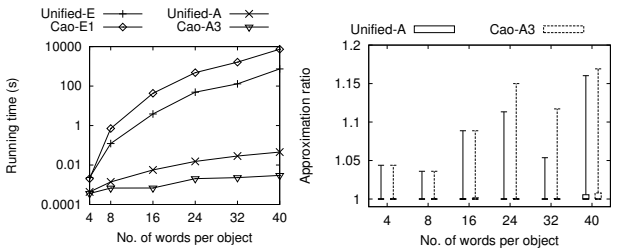
(3) $cost_{Sum}$**.** The results for $cost_{Sum}$ are shown in Figure 11. According to the Figure 11(a), *Unified-E* runs slower than *Cao-E2*, and the reason is perhaps that the pruning technique of *Unified-E* based on dominant objects becomes less effective when $|o.\psi|$ increases. Besides, *Unified-A* runs slightly slower than *Cao-A3* but gives a better approximation than *Cao-A3* (Figure 11(b)). This is because *Unified-A* construct a feasible set for each key query-object distance contributor and pick the best one as the solution.



(a) Running time (b) Approximation ratio

Figure 11: Effect of average $|o.\psi|$ on $cost_{Sum}$

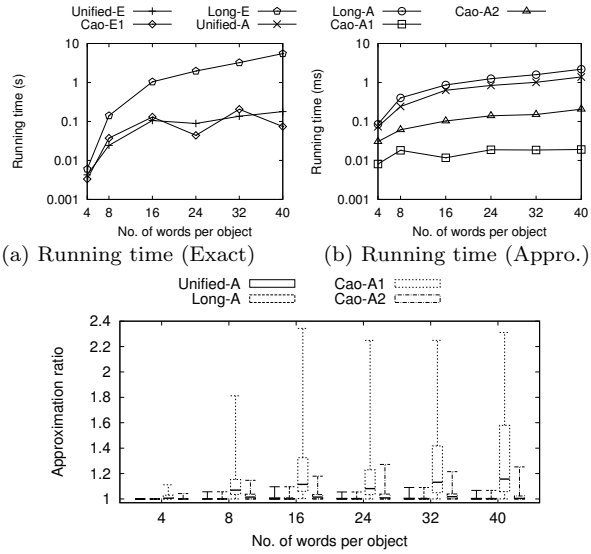
(4) $cost_{SumMax}$**.** Under the default setting of $|q.\psi| = 10$, the running times of all exact algorithms including *Unified-E* and *Cao-E1* grow very rapidly when $|o.\psi|$ increases, e.g., the algorithms ran for more than 1 day when $|o.\psi| \geq 8$. Thus, for better comparison among the algorithms, we particularly use the setting of $|q.\psi| = 8$ for $cost_{SumMax}$. According to Figure 12(a), *Unified-E* runs consistently faster than *Cao-E1* and *Unified-A* runs fast, though not as fast as *Cao-A3*, and gives a better approximation ratio (Figure 12(b)). Specifically, the largest approximation ratios of *Unified-A* and *Cao-A3* are 1.160 and 1.169, respectively.



(a) Running time (b) Approximation ratio

Figure 12: Effect of average $|o.\psi|$ on $cost_{SumMax}$

(5) $cost_{MaxMax}$**.** The results for $cost_{MaxMax}$ are shown in Figure 13. According to Figure 13(a), *Unified-E* is one of the two algorithms that run the fastest and the other is *Cao-E1*. According to Figure 13(b) and (c), all approximate algorithms including *Unified-A* run reasonably fast and *Unified-A* is one of the two algorithms which give the best approximation ratios (the other is *Long-A*). Specifically, the largest approximation ratios of *Unified-A* is only 1.096, while that of *Cao-A1* and *Cao-A2* are 2.343 and 1.271, respectively, which are much larger.



(a) Running time (Exact) (b) Running time (Appro.)

(c) Approximation ratio

Figure 13: Effect of average $|o.\psi|$ on $cost_{MaxMax}$

(6) $cost_{MaxMax2}$**.** The results for $cost_{MaxMax2}$ are shown in Figure 14, which are similar to those for $cost_{MaxMax}$, i.e., *Unified-E* is one of the two fastest exact algorithm and *Unified-A* runs reasonably fast and is one of the two algorithms which give the best approximation ratios.

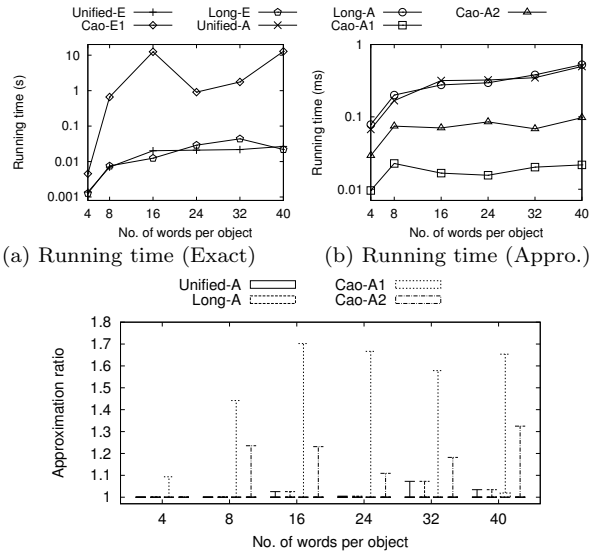
(7) $cost_{Max}$**.** The results for $cost_{Max}$ are shown in Figure 15(b). According to the results, both *Unified-E* and *Unified-A* run very fast, e.g., they ran less than 0.02 ms on all settings of $|o.\psi|$.

5.2.3 Scalability Test

Following the existing studies [3, 17, 2], we conducted experiments on scalability. According to the results, we know that both *Unified-E* and *Unified-A* are scalable to large datasets. Due to the page limit, we put the details in [6].

5.2.4 Summary Of Experimental Results

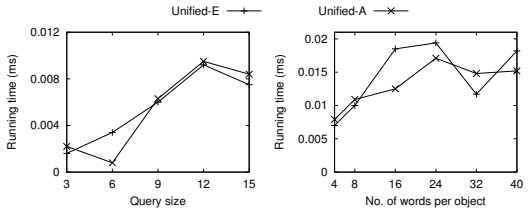
Our exact algorithm *Unified-E* is clearly the best exact algorithm for CoSKQ queries not only because it is a *unified*



(a) Running time (Exact) (b) Running time (Appro.)

(c) Approximation ratio

Figure 14: Effect of average $|o.\psi|$ on $cost_{MaxMax2}$



(a) Effect of $|q.\psi|$ (b) Effect of average $|o.\psi|$

Figure 15: Experiments on $cost_{Max}$

approach but also it is always among those with the best running times (e.g., it beats the state-of-the arts *consistently* for $cost_{MinMax}$ and $cost_{MinMax2}$, *when* $|q.\psi|$ *becomes large* for $cost_{Sum}$ and $cost_{SumMax}$, and *under the majority of settings* for $cost_{MaxMax}$ and $cost_{MaxMax2}$).

Our approximate algorithm *Unified-A* runs reasonably fast (e.g., for the majority settings of $|q.\psi|$, it ran within 0.1 seconds), while sometimes it is not as fast as the competitors because *Unified-A* has some more checking so that it can take care all cost functions. Meanwhile, *Unified-A* is always among the those which give the best approximation ratios close to 1 and runs always faster than those algorithms which give similar approximation ratios as *Unified-A*.

6. CONCLUSION

In this paper, we proposed a unified cost function for CoSKQ. This cost function expresses all existing cost functions in the literature and a few cost functions that have not been studied before. We designed a unified approach, which consists of one exact algorithm and one approximate algorithm. The exact algorithm runs comparably fast as the existing exact algorithms, while the approximate algorithm provides a comparable approximation ratio as the existing approximate algorithms. Extensive experiments were conducted which verified our theoretical findings.

There are several interesting future research directions. One direction is to design a cost function such that it penalizes those objects with too much keywords for fairness. It is also interesting to extend the unified approach to handle the route-oriented spatial keyword queries. Besides, it is left as a remaining issue to study the CoSKQ problem with a moving query point.

7. REFERENCES

[1] X. Cao, L. Chen, G. Cong, and X. Xiao. Keyword-aware optimal route search. *PVLDB*, 5(11):1136–1147, 2012.

[2] X. Cao, G. Cong, T. Guo, C. S. Jensen, and B. C. Ooi. Efficient processing of spatial group keyword queries. *TODS*, 40(2):13, 2015.

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

[4] X. Cao, G. Cong, C. S. Jensen, and M. L. Yiu. Retrieving regions of intersect for user exploration. *PVLDB*, 7(9), 2014.

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

[6] H. K.-H. Chan, C. Long, and R. C.-W. Wong. On generalizing collective spatial keyword queries (full version). In *http://www.cse.ust.hk/~khchanak/paper/coskq-full.pdf*, 2017.

[7] D.-W. Choi, J. Pei, and X. Lin. Finding the minimum spatial keyword cover. In *ICDE*, pages 685–696. IEEE, 2016.

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

[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] K. Deng, X. Li, J. Lu, and X. Zhou. Best keyword cover search. *TKDE*, 27(1):61–73, 2015.

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

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

[13] Y. Gao, J. Zhao, B. Zheng, and G. Chen. Efficient collective spatial keyword query processing on road networks. *ITS*, 17(2):469–480, 2016.

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

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

[16] J. Liu, K. Deng, H. Sun, Y. Ge, X. Zhou, and C. Jensen. Clue-based spatio-textual query. *PVLDB*, 10(5):529–540, 2017.

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

[18] J. Rocha, O. Gkorgkas, S. Jonassen, and K. Nørvåg. Efficient processing of top-k spatial keyword queries. *SSTD*, pages 205–222, 2011.

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

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

[21] A. Skovsgaard and C. S. Jensen. Finding top-k relevant groups of spatial web objects. *VLDBJ*, 24(4):537–555, 2015.

[22] S. Su, S. Zhao, X. Cheng, R. Bi, X. Cao, and J. Wang. Group-based collective keyword querying in road networks. *Information Processing Letters*, 118:83–90, 2017.

[23] D. Wu, G. Cong, and C. Jensen. A framework for efficient spatial web object retrieval. *VLDBJ*, 21(6):797–822, 2012.

[24] D. Wu, M. Yiu, G. Cong, and C. Jensen. Joint top-k spatial keyword query processing. *TKDE*, 24(10):1889–1903, 2012.

[25] 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.

[26] Y. Zeng, X. Chen, X. Cao, S. Qin, M. Cavazza, and Y. Xiang. Optimal route search with the coverage of users' preferences. In *IJCAI*, pages 2118–2124, 2015.

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

[28] 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.

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

[30] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. EDBT/ICDT, pages 359–370. ACM, 2013.