Cost-Aware and Distance-Constrained Collective Spatial Keyword Query

Chan, Harry Kai Ho; Liu, Shengxin; Long, Cheng; Wong, Raymond Chi Wing

Published in:
IEEE Transactions on Knowledge and Data Engineering

DOI:
10.1109/TKDE.2021.3095388

Publication date:
2023

Document Version
Peer reviewed version

Citation for published version (APA):
Chan, H. K. H., Liu, S., Long, C., & Wong, R. C. W. (2023). Cost-Aware and Distance-Constrained Collective Spatial Keyword Query. *IEEE Transactions on Knowledge and Data Engineering*, *35*(2), 1324-1336. https://doi.org/10.1109/TKDE.2021.3095388

Cost-Aware and Distance-Constrained Collective Spatial Keyword Query

Harry Kai-Ho Chan, Shengxin Liu, Cheng Long, and Raymond Chi-Wing Wong

Abstract—With the proliferation of location-based services, geo-textual data is becoming ubiquitous. Objects involved in geo-textual data include geospatial locations, textual descriptions or keywords, and various attributes (e.g., a point-of-interest has its expenses and users' ratings). Many types of spatial keyword queries have been proposed on geo-textual data. Among them, one prominent type is to find, for a query consisting of a query location and some query keywords, a set of multiple objects such that the objects in the set collectively cover all the query keywords and the object set is of good quality according to some criteria. Existing studies define the criteria either based on the geospatial information of the objects solely or simply treat the geospatial information and the attribute information of the objects together without differentiation though they may have different semantics and scales. As a result, they cannot provide users flexibility to express finer grained preferences on the objects. In this paper, we propose a new criterion which is to find a set of objects where the distance (defined based on the geospatial information) is at most a threshold specified by users and the cost (defined based on the attribute information) is optimized. We develop a suite of two algorithms including an exact algorithm and an approximation algorithm with provable guarantees for the problem. We conducted extensive experiments on real datasets which verified the efficiency and effectiveness of proposed algorithms.

Index Terms—Spatial keyword queries, spatial database, query processing

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 points-of-interests (POIs) in the physical world (e.g., restaurants, shops and hotels), geo-tagged web objects (e.g., webpages and photos at Flicker), and geo-social networking data (e.g., users of FourSquare have their check-in histories which are spatial and also have their profiles which are textual). Entities of geo-textual data, which we call geo-textual objects, are associated with various attributes as well. For example, a restaurant is usually associated with some expense attribute (e.g., in the Yelp APP, this information is shown by the number of "$" symbols), a tourism site would require the admission fee, and most point-of-interests (POIs) have their popularity reflected by users' ratings.

Many types of queries have been proposed on a database of geo-textual objects [16], [6], [31], [38], [35], [4], [21], [3], [7], [8], where a query usually consists of a location called the query location (e.g., the current location of a user who issues the query) and some keywords called query keywords (e.g., those which express the interests of a user who issues the query). Among these queries, a popular one is to search for a set of multiple objects that collectively cover all query keywords and are desirable to the user based on some criterion [4], [21], [3], [7], [8]. For example, a tourist who wants to do sight-seeing, shopping, and dining could issue such a query with a location such as his/her current location as the query location and keywords "sigh-seeing", "shopping", and "dinning" as the query keywords, and the query would return to the tourist a few POIs which collectively satisfy the tourist's needs and also be of high quality according to some criterion (e.g., the POIs are physically close to the query location). If an object set covers all query keywords, such object set is said to be a *feasible set*. There are usually many feasible sets given a query - each combination of objects covering different query keywords would be a feasible set. Therefore, a key question that needs to be answered is that among all possible feasible sets, which one should be returned, i.e., what criterion should be used for picking a feasible set?

Most studies define the criterion based on the geospatial aspects of the objects solely [4], [21], [3], [8]. Specifically, the criterion is to find the feasible set which has the smallest "distance" wrt the query location, where the distance of a set of objects wrt the query location is defined based on the distances between the objects and the query location and also those among the objects. For example, in [4], [21], the distance of a set of objects wrt a query location is defined as the sum of the maximum distance between an object and the query location and the maximum distance between two objects. While using this criterion would help to find a feasible set with some notation of distance optimized, it pays no attention to the attribute aspect of the objects which results in no guarantee on the desirability of the returned feasible set in the attribute aspect. For example, it

- *H.K.-H. Chan is with the Department of People and Technology, Roskilde University, Denmark. E-mail: kai-ho@ruc.dk (Partial of the work was done at Nanyang Technological University as a visiting PhD student)*
- *S. Liu is with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. E-mail: sxliu@hit.edu.cn*
- *C. Long is with the School of Computer Science and Engineering, Nanyang Technological University, Singapore. E-mail: c.long@ntu.edu.sg*
- *R.C.-W. Wong is with the Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: raywong@cse.ust.hk*

Manuscript received 10 Mar. 2020; revised 8 May, 2021; accepted 24 Jun., 2021 (Corresponding Authors: Shengxin Liu and Cheng Long)

may happen that a user who prefers restaurant with low or moderate prices is returned with a restaurant with a super high price simply because the restaurant is close to the user.

A more recent study defines the criterion such that it considers both the geospatial aspect and the attribute aspect of objects [7]. Specifically, it interprets an object's attribute as a form of "cost" (e.g., the expense of a restaurant is interpreted as monetary cost, the popularity of a shop can be captured by $cost = e^{-score}$ where $score \in [0, 1]$ can be a user rating) and the criterion is defined to find the feasible set with the smallest product between a "cost" and a "distance", where the cost of an object set is defined as the maximum cost of an object and the distance is defined similarly as previous studies [4], [21], [3], [8] that use distances solely. With this criterion, the query would tend to find a feasible set with a small cost and also a small distance. While this criterion is superior over previous ones that ignore the cost part, it lacks of capability of providing users a finer grained control on their preferences on the cost part and the distance part. Specifically, this criterion simply mixes the two parts together with the product operator and gives no interface for users to express which part they care about more and which less. For example, a common scenario is that a user would want to find a feasible set whose "distance" is within a range, e.g., 1km, and has its "cost" as small as possible, and yet this cannot be handled by the criterion in [7]. Moreover, a cost and a distance may have different semantics and scales, and combining them together using a product operator may cause problems such that one dimension dominates the other and essentially only one aspect is captured.

Motivated by the above discussion, in this paper, we propose a new criterion which is to find a feasible set such that its distance is at most a pre-defined threshold and its cost is as small as possible. Our rationale is that (1) using the distance and the cost separately would give more flexibility for users to express their preferences and (2) imposing a constraint on the distance and optimizing the cost would capture more use cases since after all a user cares about the cost more as long as the distance is being acceptable (i.e., being below a threshold). We call the resulting query the *Cost-Aware and Distance-Constrained Collective Spatial Keyword Query* (CD-CoSKQ). Same as [7], the attribute aspect of an object is interpreted as the cost of the object, and the cost of an object set corresponds to an aggregated form of the costs of the objects. Depending on the semantic of the attribute, the costs of objects should be aggregated in different ways. For example, in the case where the attribute corresponds to some expense, the costs should be better aggregated using the *sum* function while in the case where the attribute corresponds to some dissatisfaction reflected by ratings, the costs should be better aggregated using the *max* function. Therefore, in CD-CoSKQ, we consider two cost definitions, namely $cost_{Sum}(\cdot)$ and $cost_{Max}(\cdot)$, where $cost_{Sum}(\cdot)$ defines the cost of an object set as the sum of the costs of the objects and $cost_{Max}(\cdot)$ as the maximum cost of an object. Regarding the distance of an object set, some distance functions have been proposed in the literature of collective spatial keyword query, and in this paper, we consider two of them, namely $dist_{Dia}(\cdot)$ [21], [8] and $dist_{MaxSum}(\cdot)$ [4], [3], [21], [8] and leave others for future

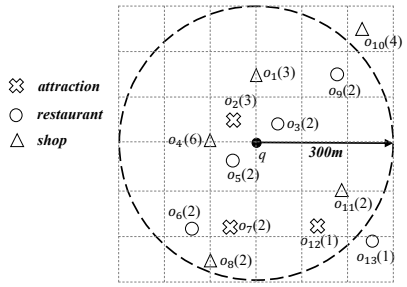


Fig. 1. Running example

exploration. Nevertheless, the algorithmic framework is applicable to other cost functions that are monotonic increasing, and extensible to other distance functions (e.g., $dist_{MinMax}(\cdot)$ [3], $dist_{SumMax}(\cdot)$ [8] and $dist_{Max}(\cdot)$ [8]) in the literature, since these distance functions include the maximum distance between two objects, and our framework utilizes this property to identify candidate set regions that contain the optimal solution.

Consider a user who wants to buy some medicine, a computer mouse and a coffee within her/his walking distance. S/he could issue a CD-CoSKQ with query keywords *medicine*, *mouse* and *coffee* and a distance threshold of 300m. When $dist_{Dia}(\cdot)$ and $cost_{Sum}(\cdot)$ are adopted, the query should return a feasible set with the minimum total cost, among all feasible sets that satisfy the distance constraint (i.e., $dist_{Dia}(\cdot) \leq 300m$).

To further illustrate the advantages of CD-CoSKQ over the existing CoSKQ formulations, consider the example in Figure 1, where we use $cost_{Sum}(G) = \sum_{o \in G} o.w$ for the cost definition and $dist_{Dia}(G) = \max_{o_1, o_2 \in G \cup \{q\}} d(o_1, o_2)$ as the distance function. Figure 1 shows the example with 13 objects and a query location, where the value in the parentheses is the cost of each object. Suppose the query keywords are $attraction$, $restaurant$ and $shop$, and the distance threshold B = 300m. We compare the optimal solution for different formulations of CoSKQ, as listed in Table 1. In particular, we introduce a weighted MaxDotSize-CoSKQ, which generalizes the cost function in [7] to $cost^{w_1} \times distance^{w_2}$, where $w_1, w_2 \in [0, 1]$ allow users to control the weights for the two parts. In this example, we set $w_1 = 1$ and $w_2 = 0.5$ to relax the constraint in distance part. The optimal solution of our CD-CoSKQ has a cost of 6 only, which is the smallest among all formulations, while its distance is still within the query distance threshold. This shows that CD-CoSKQ is able to find a solution that is more desirable to user in this case.

TABLE 1
Optimal solutions of different CoSKQ formulations

	Optimal Solution G	$dist_{dia}(G)$	$cost_{Sum}(G)$
Dia-CoSKQ [21]	$\{o_2, o_4, o_5\}$	100m	11
MaxDotSize-CoSKQ [7]	$\{o_2, o_4, o_5\}$	100m	11
Weighted MaxDotSize-CoSKQ	$\{o_1, o_2, o_3\}$	150m	8
CD-CoSKQ (this work)	$\{o_6, o_7, o_8\}$	269m	6

In general, the CD-CoSKQ problem is proven to be NP-hard with respect to the number of query keywords and even NP-hard to approximate with constant factors, as we will prove in Theorem 1 shortly. Yet for queries with a small number of query keywords, performing a CD-CoSKQ within a reasonable amount of time is quite doable (e.g., in the case a query involves two query keywords only, one can

solve the problem exactly by checking all pairs of objects covering the query keywords and the time complexity is quadratic). Therefore, we first develop an exact algorithm called *CD-Exact*, which uses a carefully designed search strategy, employs various pruning techniques, and runs based on the widely used IR-tree [11] augmented with some cost information. For problems that are NP-hard to approximate, a popular approach in the literature [22], [15], [1] is to develop a bi-criteria approximation algorithm. Thus, we develop an (α, β)-approximation algorithm *CD-Appro*, where the returned object set has its cost within a factor of α from the minimum cost of any feasible set satisfying the distance threshold constraint, and its distance at most β times the distance threshold. For different cost functions and distance functions, α and β are different. For example, when $cost_{Max}(\cdot)$ and $dist_{MaxSum}(\cdot)$ are used, CD-Appro is a $(1, 1.375)$-approximation algorithm.

In summary, our main contribution is summarized as follows.

- First, we propose a new type of query, namely CD-CoSKQ, which aims to find an object set with the smallest cost subject to a constraint on the distance. The cost is based on the attribute aspect of the objects and the distance on the geospatial aspect. This query provides users a finer grained interface to express their preferences on both the geospatial aspect and the attribute aspect of the geo-textual objects.
- Second, we prove the inapproximability result of the CD-CoSKQ and develop a suite of two algorithms for the query, namely an exact algorithm *CD-Exact* and an (α, β)-approximation algorithm *CD-Appro*.
- Third, we conducted extensive experiments on both real and synthetic datasets, which verified our theoretical results and the efficiency of our algorithms.

The rest of this paper is organized as follows. Section 2 defines the problem formally and discusses its hardness. Sections 3, 4 and 5 present our algorithms. Section 6 gives the empirical study. Section 7 reviews the related work and Section 8 concludes the paper.

2 Problem Definition

Let $\mathcal{O}$ be a set of geo-textual objects. Each object $o \in \mathcal{O}$ is associated with a location denoted by $o.\lambda$, a set of keywords denoted by $o.\psi$, and some attributes which we convert to a form of cost denoted by $o.w$ such that a lower cost is preferred. For example, for an attribute of expense, we convert it to a cost equal to the expense and for an attribute of dissatisfaction such as a rating, we convert it to a cost which is inverse to the attribute. Given two objects o and o', we denote by $d(o, o')$ the Euclidean distance between $o.\lambda$ and $o'.\lambda$.

The main notations that are used throughout the paper are summarized in Table 2.

TABLE 2
Notation table

Notation	Definitions
$\mathcal{O}$	a set of geo-textual objects
$\mathcal{O}(t)$	a set of geo-textual objects each of which contains the keyword t
q	a query with location $q.\lambda$, a set of keywords $q.\psi$, and a distance threshold $q.B$
t_{inf}	the query keyword which has the smallest number of objects containing it
G	a (feasible) set of objects
S	a candidate set of objects
G^*	the optimal solution of the query
S^*	the candidate set that is the superset of G^*
$D(q, r)$	the disk with its center at location of q and its radius r

2.1 Problem Definition

We formally define the Cost-Aware and Distance-Constrained Collective Spatial Keyword Query (CD-CoSKQ) problem as follows.

Problem 1 (CD-CoSKQ). Let $\mathcal{O}$ be a set of geo-textual objects. Given a query q which consists of a query location $q.\lambda$, a set of query keywords $q.\psi$, and a distance threshold $q.B$, the CD-CoSKQ problem is to find a set $G \subseteq \mathcal{O}$ of objects such that (1) G covers $q.\psi$, (2) the distance of G wrt q, denoted by $dist(G, q)$, is at most $q.B$, and (3) the cost of G, denoted by $cost(G)$, is minimized.
Formally, the problem is to find the optimal solution G^* defined as follows.

$$\begin{array}{ll} G^* = \arg\min_{G \subseteq \mathcal{O}} & cost(G) \\ \text{s. t.} & q.\psi \subseteq \cup_{o \in G} o.\psi, \\ & dist(G, q) \leq q.B, \end{array} \quad (1)$$

We simply write the distance of G as $dist(G)$ and the distance threshold $q.B$ as B when the context of q is clear. □

Given a query q, an object is said to be *relevant* if it contains at least one keyword in $q.\psi$ and a set S of objects is said to be a *feasible set* if S covers $q.\psi$ (i.e., $q.\psi \subseteq \cup_{o \in S} o.\psi$). Thus, the CD-CoSKQ problem is to find a feasible set which has its distance at most a threshold and its cost as small as possible. In practice, the value of the distance threshold B could be a real value (e.g., 1km), or in case that the users do not have a concrete idea on how to determine the threshold, one could allow to set a factor n, which is the multiple that the user can tolerance compared to the result of the conventional CoSKQ. As shown in our experiment, n = 1.1 is a reasonable value for different datasets.

Cost Functions. In this paper, we consider two cost functions $cost(G)$ as follows.

$$cost_{Max}(G) = \max_{o \in G} o.w \quad (2)$$

$$cost_{Sum}(G) = \sum_{o \in G} o.w \quad (3)$$

$cost_{Max}(G)$ aggregates the costs of the objects in G as the worst-case cost of an object in G and is suitable for cases where the costs of the objects correspond to some forms of dissatisfaction such as those based on ratings and the worst-case cost reflects the tolerable level of the cost of the object set. $cost_{Sum}(G)$ aggregates the costs of the objects in G using a sum function and suitable for cases where the costs of the objects correspond to some form of expenses such as time and money and thus, the cost of an object set intuitively corresponds to the sum of the costs of the objects.

Distance Functions. We consider two commonly used functions, namely $dist_{MaxSum}(G)$ [4], [3], [21], [8] and $dist_{Dia}(G)$ [21], [8], as the distance function in this paper.

$$dist_{MaxSum}(G) = \max_{o \in G} d(o, q) + \max_{o_1, o_2 \in G} d(o_1, o_2) \quad (4)$$

$$dist_{Dia}(G) = \max_{o_1, o_2 \in G \cup \{q\}} d(o_1, o_2) \quad (5)$$

Both $dist_{MaxSum}(G)$ and $dist_{Dia}(G)$ define the distance of G based on two distances, namely the maximum distance between an object in G and q, and the maximum distance between two objects in G. $dist_{MaxSum}(G)$ uses the sum of the two distances while $dist_{Dia}(G)$ uses the maximum of them. As explained in previous studies [4], [21], these two distance functions suit different needs. Note that other distance functions such as those proposed by [4], [8] could also be used here, and yet due to the page limit, we leave them for future explorations.

Intractability. We show that the CD-CoSKQ problem is NP-hard to approximate with any constant factor c ($c \geq 1$), i.e., there does not exist any polynomial-time algorithm which can decide whether there exists a solution to the problem and if so, return a solution with its cost at most c times that of the optimal one unless P = NP. Note that it immediately follows that the problem is NP-hard.

Theorem 1. The CD-CoSKQ problem is NP-hard to approximate with any constant factor c ($c \geq 1$). □

Proof: The proof could be found in Appendix A, available online. □

2.2 Indexing

Following the existing studies on spatial keyword queries [3], [21], [8], in this paper, we adopt the IR-tree [11] for keyword-based nearest neighbour queries and range queries which are procedures invoked in the algorithms to be introduced in this paper. The conventional IR tree augments an R-tree by storing at each node an inverted list which maintains for each keyword those children nodes which store an object containing the keyword. To suit our algorithms better, we augment the standard IR-tree by including some extra cost information in each inverted list which will be used for pruning. Specifically, in each inverted list of a keyword, we maintain not only the children nodes which store an object containing the keyword but also the minimum cost of these objects that are stored in the node and contain the keyword.

3 Exact Algorithm

In this section, we introduce our exact algorithm CD-Exact. For the ease of presentation, we focus on the cost function $cost_{Max}$ (Equation 2) and the distance function $dist_{MaxSum}$ (Equation 4) first (in Section 3 and 4) and then discuss how the algorithms could be used to handle the cost function $cost_{Sum}$ (Equation 3) and the distance function $dist_{Dia}$ (Equation 5) (in Section 5).

Given a query q and a keyword t, the **t-keyword minimum cost neighbor** of q, denoted by $MN(q, t)$, is defined to be the object located within $D(q, B)$ containing the keyword t with the minimum cost. We define the **minimum cost neighbor set** of q, denoted by $M(q)$, to be the set containing q's t-keyword minimum cost neighbor for each $t \in q.\psi$, i.e., $M(q) = \cup_{t \in q.\psi} MN(q, t)$. Note that $M(q)$ is a feasible set.

Consider the optimal solution G^* for a given CD-CoSKQ query q. By definition, G^* is a feasible set, i.e., for each query keyword, G^* must include an object containing this keyword. Let t be a query keyword and $\mathcal{O}(t)$ be the set of objects, each of which contains t. We know that G^* must include an object in $\mathcal{O}(t)$. Motivated by this, we propose to find G^* by searching around each object $o \in \mathcal{O}(t)$, which we call a **seed object**. Specifically, to search around an object $o \in \mathcal{O}(t)$, we restrict our attention to a set S of relevant objects that are located close enough to o wrt the query distance threshold, which we call a **candidate set**, and then within S, we find the feasible set G with the smallest cost among all feasible sets involving o and having the distance at most B, which we call a **local optimal set**. At the end, we return the local optimal set which has the smallest cost among all local optimal sets found, and it is deemed to be the optimal solution.

Based on the above discussion, we design an algorithm called *CD-Exact*, which involves four steps as follows.

- **Step 1 [Seed Object Exploration]**: Find an object o from $\mathcal{O}(t)$ that has not been explored as a seed object.
- **Step 2 [Candidate Set Construction]**: Construct a candidate set S of objects based on the seed object o.
- **Step 3 [Local Optimal Set Finding]**: Find the local optimal set G within S and update the best known local set, denoted by $G_{curBest}$, if necessary.
- **Step 4 [Iterative Step]**: Repeat Step 1 to Step 3 until all objects in $\mathcal{O}(t)$ have been explored, in which case, return $G_{curBest}$.

In the above algorithm, $G_{curBest}$ is initialized to be an empty set $\varnothing$ at the beginning. If $dist_{MaxSum}(M(q)) \leq B$, we initialize $G_{curBest}$ to $M(q)$. At the end, if $G_{curBest}$ remains to be $\varnothing$, it implies that there does not exist a solution for the query; otherwise, $G_{curBest}$ corresponds to the optimal solution (proof will be provided shortly). The pseudo-code of CD-Exact is presented in Algorithm 1.

Next, we present details of Steps 1-3 (note that Step 4 is straightforward), prove the correctness of the algorithm, and give the complexity analysis on the algorithm.

Step 1 [Seed Object Exploration]. Seed objects, i.e., those in $\mathcal{O}(t)$, are defined based on a query keyword. Using different query keywords, we would have different sets of seed objects. Considering that the number of seed objects directly implies the number of iterations in the algorithm, we use the query keyword such that the corresponding set of seed objects is the smallest. In other words, we use the query keyword which has its frequency measured by the number of objects containing it the lowest. We denote this query keyword by t_{inf}. That is, we explore all objects in $\mathcal{O}(t_{inf})$ as seed objects. Note that we follow [3] that we preprocess the dataset to compute the frequency of each keyword.

Moreover, since the goal is to find the optimal solution G^*, whose distance is bounded by the distance threshold B, we can safely ignore those seed objects o whose distances

Algorithm 1 CD-Exact

Input: An object set $\mathcal{O}$, a query q with a location $q.\lambda$, a set of keywords $q.\psi$, and a distance threshold $q.B$
Output: A feasible set with the smallest cost and having the distance at most $q.B$

1: $G_{curBest} \leftarrow \emptyset$; $curCost \leftarrow \infty$
2: **if** $dist_{MaxSum}(M(q)) \leq q.B$ **then**
3: $\quad G_{curBest} \leftarrow M(q)$; $custCost \leftarrow cost_{Max}(M(q))$
4: // **Step 1 [Seed Object Exploration]**
5: $t_{inf} \leftarrow$ the most infrequent keyword in $q.\psi$
6: **for** each object o in $D(q, q.B)$ and contains t_{inf} **do**
7: $\quad$ // **Step 2 [Candidate Set Construction]**
8: $\quad R(o) \leftarrow D(o, q.B - d(o,q))$
9: $\quad S \leftarrow$ the set of all relevant objects o' in $R(o)$ with $o'.w < curCost$
10: $\quad$ // **Step 3 [Local Optimal Set Finding]**
11: $\quad \psi \leftarrow q.\psi \setminus o.\psi$
12: $\quad$ **if** S does not cover ψ **then**
13: $\quad\quad$ **continue**
14: $\quad$ **else if** $\psi = \emptyset$ **then**
15: $\quad\quad G \leftarrow \{o\}$
16: $\quad$ **else**
17: $\quad\quad G \leftarrow$ localOptimalSetFinding(o, S, ψ)
18: $\quad$ **if** $G \neq \emptyset$ *and* $cost_{Max}(G) \leq curCost$ **then**
19: $\quad\quad G_{curBest} \leftarrow G$; $curCost \leftarrow cost_{Max}(G)$
20: $\quad$ // **Step 4 [Iterative Step]**
21: **return** $G_{curBest}$

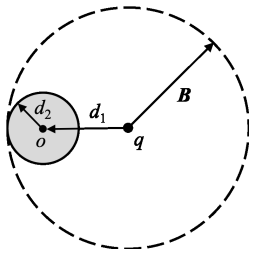


Fig. 2. A candidate set region in CD-Exact

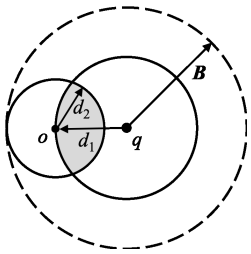


Fig. 3. A candidate set region in CD-Appro

from q are greater than B since their corresponding local optimal sets (which includes o) have their distance greater than B and thus they cannot be the optimal solution. In summary, we only need to consider those objects that (1) contain t_{inf} and (2) are located in $D(q, B)$ as seed objects (Lines 3 - 4 in Algorithm 1).

Step 2 [Candidate Set Construction]. Consider a seed object o containing t_{inf}. The purpose of this step is to construct a local region near o, denoted by $R(o)$, such that the local optimal set G based on o is inside the region. Let $d_1 = d(o, q)$ and $d_2 = B - d_1$. We construct the region $R(o)$ as $D(o, d_2)$, i.e., the disk with its center as o and its radius as d_2 as shown in the gray region in Figure 2. Then, the set of relevant objects inside the region $R(o)$ corresponds to a candidate set S. With $R(o)$ constructed in this way, we guarantee that the local optimal set G based on o is inside $R(o)$ (i.e., $G \subseteq S$).

Lemma 1. The local optimal set G based on o is inside $R(o) = D(o, B - d(o,q))$. □

Proof: We prove by contradiction. Suppose G includes an object o' which is outside the range $R(o)$. Then, $dist_{MaxSum}(G) = \max_{o'' \in G} d(o'', q) + \max_{o_1, o_2 \in G} d(o_1, o_2) > d(o,q) + d_2 = B$, which contradicts to the fact that G has its distance at most B. □

Algorithm 2 localOptimalSetFinding(o, S, ψ)

Input: A seed object o, the candidate set S based on o, and the set ψ containing those query keywords that are not covered by o
Output: The local optimal set G based on o if any and $\emptyset$ otherwise

1: $bestG \leftarrow \emptyset$; $bestGCost \leftarrow \infty$
2: **for** each subset G' of S, which covers ψ **do**
3: $\quad G \leftarrow G' \cup \{o\}$
4: $\quad$ **if** $dist_{MaxSum}(G) \leq B$ **then**
5: $\quad\quad$ **if** $cost_{Max}(G) < bestGCost$ **then**
6: $\quad\quad\quad bestGCost \leftarrow cost_{Max}(G)$; $bestG \leftarrow G$
7: **return** $bestG$

Furthermore, we can refine the candidate S by dropping all those objects with the cost greater than $curCost$ since a set of objects including any of these objects would have the cost at least that of the current best one and thus, it could be safely pruned. In summary, we construct the candidate set S based on o to be the set including all relevant objects which are in $R(o)$ and have the cost smaller than $curCost$ (Lines 6 - 7 in Algorithm 1).

Step 3 [Local Optimal Set Finding]. By definition, the local optimal set G within the candidate set S (if it exists) should include o and some other objects in S. Thus, we aim to find G by exploring all subsets of S and augmenting each of them with the object o as a candidate of G. Instead of enumerating all subsets of S, we only consider those which cover the query keywords that have not been covered by object o since any other subsets when augmented with object o cannot cover all query keywords meaning that it cannot be G. Specifically, we maintain the set of the query keywords that are not covered by o in ψ, i.e., $\psi = q.\psi \setminus o.\psi$. There are three cases. First, S as a whole does not cover ψ. In this case, we can safely conclude that the local optimal set for o does not exist and proceed to the next iteration. Second, ψ is an empty set, i.e., o covers all query keywords. In this case, we can safely return $G = \{o\}$ as the local optimal set for o. Third, it corresponds to all other cases (which are hard ones). We then invoke a procedure called "localOptimalSetFinding" (presented in Algorithm 2) which enumerates all subsets of S that cover ψ and for each one, (1) augments it with the object o; (2) checks whether the augmented set has the distance at most B and if so, (3) updates the best known augmented set $bestG$ when necessary and returns $bestG$ at the end. We then update the current best solution $G_{curBest}$ when necessary. These steps are presented in Lines 9 -17 of Algorithm 1.

Moreover, to utilize the characteristics of the $cost_{Max}$ function, we develop a few pruning techniques for the process of enumerating the subsets of S. Enhanced by these pruning techniques, CD-Exact runs significantly faster than baseline methods.

One technique is to enumerate the objects in S in an ascending order of the objects' costs, which would make it possible to terminate the search process early since the process at a later stage would only find sets of objects with the costs larger than the best known cost and thus they could be pruned. In particular, we consider the objects in S in ascending order of their costs. Let $S = \{o_1, o_2, \ldots, o_n\}$ where $o_i.w \leq o_{i+1}.w$ for all $1 \leq i \leq n-1$. For each iteration

$i \in \{1, 2, \ldots, n\}$, we only consider the combinations of the objects in the set S_i' where $S_i' = \{o_1, o_2, \ldots, o_i\}$. To enumerate the objects in S_i' efficiently, we utilize the inverted lists. In each level, we only need to consider objects in the lists that contain keywords that are not covered yet. The advantage of this search strategy is that we can terminate the search immediately once we found any feasible solution G within distance threshold, since the remaining objects (which has higher costs than the objects we have processed according to the ascending order) in S cannot contribute to a better solution. In more details, let's consider an example. Suppose that we find a feasible solution sol_i in S_i' and another feasible solution sol_j in S_j' with $i < j$. Based on the order in S, we clearly know that sol_i has smaller cost than sol_j. In other words, once we can get a feasible solution with S_i', we do not need to consider any cases with S_j' where $j \geq i$.

Another pruning technique is that we may start finding the local optimal solution from $S_j' = \{o_1, o_2, \ldots, o_j\}$ for some $j > 1$ instead of $j = 1$. In particular, we impose the following two lower bounds on $cost_{Max}(G)$, which implies that we can start the finding on S_i' where i is the smallest index such that $cost_{Max}(G) \geq o_i.w$. First, since the seed object o must be included in the constructed set G according to CD-Exact, we know that $cost_{Max}(G) \geq o.w$. Second, we can utilize the following lemma.

Lemma 2. $cost_{Max}(G^*) \geq \max_{t \in q.\psi} \min_{o|t \in o.\psi} o.w$. □

Proof: Each keyword in $q.\psi$ must be covered by at least one object. Suppose that o_{max} is the object with the maximum weight in G^* and o_{max} contains some keyword t_{max}. It is clear that we have $o_{max}.w \geq \min_{o|t_{max} \in o.\psi} o.w$ which implies the statement of the lemma. □

Based on the above lower bounds on $cost_{Max}(G^*)$, we can start the searching on S_i' where i is the smallest index such that $cost_{Max}(G) \geq \max\{o.w, \max_{t \in q.\psi} \min_{o|t \in o.\psi} o.w\}$.

Correctness. There are two cases. First, no feasible solutions exist for the query, e.g., due to a strict constraint on the distance. In this case, each time we try to search for the local optimal set for a seed object, we would obtain an empty set, and as a result, the CD-Exact algorithm would return an empty set at the end, indicating that no feasible solution exists. Second, there exist some feasible solutions, i.e., G^* exists. In this case, G^* includes an object that covers the query keyword t_{inf}, says o, which would be explored by the CD-Exact algorithm as a seed object. In the iteration of exploring o, the local optimal set, which corresponds to G^* (based on Lemma 1), would be found.

Time complexity. Let $\mathcal{O}'$ be the set of objects that contain t_{inf} in $D(q, B)$. Note that $|\mathcal{O}'| << |\mathcal{O}|$. For each object $o \in \mathcal{O}'$, we perform a range query which cost $O(\log|\mathcal{O}| + |S|)$, where $|S| << |\mathcal{O}|$ corresponds to the number of objects returned by the range query. Thus, the time complexity of CD-Exact is $O(|\mathcal{O}'| \cdot (\log|\mathcal{O}| + |S|^{|q.\psi|}))$, where $|S|^{|q.\psi|}$ is the time cost of the "localOptimalSetFinding" procedure (Algorithm 2).

4 Bi-criteria Approximation Algorithm

As we proved in Theorem 1, the CD-CoSKQ problem cannot be approximated within any constant factor in polynomial time unless $P = NP$. In this section, we present our bi-criteria approximation algorithm CD-Appro. The high-level idea is that the algorithm would output a solution G with $dist(G) \leq \beta B$ and the cost is at most α times that of the optimal solution G^*, where $dist(G^*) \leq B$. Formally, the definition of a bi-criteria approximation algorithm is shown as follows.

Definition 1. For real values $\alpha, \beta \geq 1$, an (α, β) bi-criteria approximation algorithm for CD-CoSKQ returns in polynomial time a feasible object set G such that (1) $cost(G) \leq \alpha \cdot cost(G^*)$ and (2) $dist(G) \leq \beta \cdot dist(G^*) \leq \beta \cdot B$. □

We use the terms *cost ratio* and *distance ratio* to represent the approximation ratios α and β, respectively. For both ratios, the smaller the value is, the better an algorithm performs.

CD-Appro follows CD-Exact with the following adaptions. First, it uses the set of all relevant objects in $Disk(q, B)$ as the set of seed objects (in Step 1). Second, it constructs a candidate set S for a seed object o differently as the set containing all relevant objects in the region $D(o, d_2) \cap D(q, d_1)$, as shown in the gray region of Figure 3, where $d_1 = d(o, q)$ and $d_2 = B - d_1$ (in Step 2). We abuse the notation $R(o)$ by using it to represent the region $D(o, d_2) \cap D(q, d_1)$. Third, different from CD-Exact, which performs an enumeration procedure, i.e., "localOptimalSetFinding" in Step 3, which is expensive, CD-Appro performs a greedy procedure which finds an approximate feasible set which might violate the distance constraint but has both its cost and distance from those of the one founded by "localOptimalSetFinding" bounded and runs much faster. Specifically, CD-Appro replaces the "localOptimalSetFinding" procedure with the "MaxGreedy" procedure (as shown in Algorithm 3), which finds a feasible set greedily. Given a candidate set S for some seed object and a set of query keywords ψ that are not covered by the seed object, MaxGreedy returns a set $G \subseteq S$ of objects, such that it covers ψ and $cost_{Max}(G)$ is minimized. Specifically, it checks each object o' in S in ascending order of their costs. It removes the current one if the removal would not make the set fail to cover ψ.

The intuition of CD-Appro is to construct the candidate set in a smaller region (i.e., the intersection of two disks) compared to CD-Exact, and the maximum distance between objects in the candidate set is bounded by the region. Thus, CD-Appro can enjoy a tight distance ratio without having an exhaustive search.

Note that not all relevant objects have a feasible candidate set and we can safely prune some relevant objects as

Algorithm 3 MaxGreedy(S, ψ)

Input: A candidate set S for some seed object, and the set ψ containing those query keywords that are not covered by the seed object
Output: A feasible object set G if any and ∅ otherwise
1: $G \leftarrow S$
2: **if** G does not cover ψ **then**
3: **return** ∅
4: **for** object $o' \in G$ in descending order by their costs **do**
5: **if** $G \setminus \{o'\}$ covers ψ **then**
6: $G \leftarrow G \setminus \{o'\}$
7: **return** G

Algorithm 4 SumGreedy(S, ψ)

Input: A candidate set S for some seed object, and the set ψ containing those query keywords that are not covered by the seed object
Output: A feasible object set G if any and $\varnothing$ otherwise
1: $G \leftarrow \varnothing$
2: **while** $\psi \neq \varnothing$ **do**
3: $o' \leftarrow \arg\max_{o \in S} \frac{|o.\psi \cap \psi|}{o.w}$
4: $G \leftarrow G \cup \{o'\}$
5: $\psi \leftarrow \psi \setminus o.\psi$
6: **return** G

follows. Given a query q, we find the nearest neighbors of q containing each keyword in $q.\psi$. Let o_f be the farthest object in this set of nearest neighbor objects. There exists a keyword $t_f \in o_f.\psi \cap q.\psi$ that is not contained by any object that is closer to q that o_f by definition. Given a relevant object o, we know that the corresponding candidate set S is not feasible if $d(o,q) < d(o_f,q)$ since $Disk(q, d(o,q))$ does not have any object contain t_f. Thus, we do not need to check the seed object o if $d(o,q) < d(o_f,q)$.

Approximation Ratios. We consider the cost ratio and the distance ratio separately.

Lemma 3. CD-Appro gives a cost ratio α of 1. □

Proof: Let G be the solution returned by CD-Appro. By Lemma 1, there exists an iteration in CD-Appro such that we process a set S^* that is a superset of G^*. Let G' denotes the set processed from S^* by MaxGreedy. Since CD-Appro returns the feasible set with the minimum cost, we know that $cost_{Max}(G) \leq cost_{Max}(G')$. The remaining part of the proof shows that $cost_{Max}(G') \leq cost_{Max}(G^*)$.

It is easy to see that G' covers the keywords in $q.\psi$. Let $o' \in G'$ be the object with the highest cost in G'. There exist at least one keyword in o' that contain a keyword t that is not covered by other object in G'. Consider the object $o_t \in G^*$ that contain t. We know that $cost_{Max}(G^*) \geq o_t.cost$. According to the way CD-Appro process the objects, o' is the one that has the smallest cost among all objects contain t in S^* (since otherwise $G' \setminus \{o'\}$ is feasible and o' is not in G'). Thus, $cost_{Max}(G') = o'.cost < o_t.cost \leq cost_{Max}(G^*)$. □

Lemma 4. CD-Appro gives a distance ratio β of 1.375. □

Proof: Consider a candidate set S for a seed object o. By Theorem 2 of [21], we know that $\max_{o \in S} d(o,q) + \max_{o_1,o_2 \in S} d(o_1,o_2) \leq 1.375 \cdot (d_1 + d_2)$. Thus, we have $dist_{MaxSum}(S) \leq 1.375 \cdot B$. Since the set G returned by CD-Appro (if any) is a subset of a candidate set S, we know $dist_{MaxSum}(G) \leq dist_{MaxSum}(S) \leq 1.375 \cdot B$. □

Time Complexity. Let $\mathcal{O}''$ be the set of all relevant objects that contain at least one keyword in $q.\psi$ in $D(q,B)$. Note that $|\mathcal{O}''| << |\mathcal{O}|$. The time complexity of CD-Appro is $O(|\mathcal{O}''| \cdot (\log|\mathcal{O}| + |S|\log|S| + \sum_{o \in S}|o.\psi|))$, where the part of $|S|\log|S| + \sum_{o \in S}|o.\psi|$ is the time complexity of the "MaxGreedy" procedure.

5 Extensions

5.1 Extension to the cost function $cost_{Sum}$

The algorithms for the cost function $cost_{Max}$ as described in Sections 3 and 4 remain applicable except that we need to modify the initialization (i.e., lines 2-3 of Algorithm 1 and Step 3 (i.e., the local optimal set finding step), in the algorithmic framework. The detailed modifications are shown as follows.

5.1.1 CD-Exact

In the initialization step, instead of using $M(q)$ as $G_{curBest}$ if its cost is at most B, we find a feasible set by algorithm SumGreedy (to be introduced later) using the relevant objects in $D(q,B)$ and all keywords in $q.\psi$. In Step 3, the only part that need to be adapted are that of the pruning techniques. Specifically, for $cost_{Sum}$, we terminate the search only if the current subset G' with an additional object $\arg\min_{o \in \{S \setminus G'\}} o.w$ has larger cost than $curCost$ in the "localFeasibleSetFinding" procedure. We can also do some additional pruning procedures. For example, given an object o, if the cost of $G' \cup \{o\}$ exceed $cost_{Sum}(G_{curBest})$, we do not need to add o into G'. This is because that, for any group G generated from $G' \cup \{o\}$, we have $cost_{Sum}(G) > cost_{Sum}(G_{curBest})$.

It is easy to see that these changes in the algorithm do not impair the correctness and time complexity.

5.1.2 CD-Appro

Since the cost function is changed to $cost_{Sum}$, apparently we need to design a new algorithm for finding a fairly good feasible set in Step 3 given a candidate set. Instead of calling MaxGreedy (Algorithm 3), CD-Appro invokes another algorithm, called SumGreedy (Algorithm 4), to find a feasible set greedily. Given a set S and a set of keywords ψ, SumGreedy makes use of the idea of the classic greedy algorithm for the weighted set cover where *an element* corresponds to a keyword, *a set* corresponds to an object o and the *set weight* corresponds to $o.w$.

It is easy to see that the distance ratio β of CD-Appro is not affected, since its Steps 1 and 2 are not changed. The cost ratios α and time complexity would be changed as follows.

Lemma 5. CD-Appro gives a cost ratio α of $O(\log|q.\psi|)$ for CD-CoSKQ with the cost function $cost_{Sum}$. □

Proof: Let G be the solution return by CD-Appro.. There exists an iteration in CD-Appro such that we process a set S^* that is a superset of G^*. Let G' denotes the set processed from S^* by SumGreedy. Since the algorithm returns the feasible set with the minimum cost, we know that $cost_{Sum}(G) \leq cost_{Sum}(G')$. The remaining part of the proof shows that $cost_{Sum}(G') \leq O(\log|q.\psi|) \cdot cost_{Sum}(G^*)$.

It is easy to see that G' covers the required keywords in $q.\psi$. By the well-known greedy heuristic for the weighted set cover problem [28], we know that $cost(G') \leq O(\log|q.\psi|) \cdot cost(\tilde{G})$ where $\tilde{G}$ is the optimal solution optimizing $cost_{Sum}$ (without considering any distance function). Since $cost_{Sum}(G^*)$ is the optimal solution optimizing $cost_{Sum}$ with considering a threshold B on the distance function, we have $cost_{Sum}(\tilde{G}) \leq cost_{Sum}(G^*)$. Thus we

TABLE 3
Summaries of CD-Appro

		Time Complexity	Appro. Ratio	
			α	β
$cost_{Max}$	$dist_{MaxSum}$	$O(\|\mathcal{O}''\| \cdot (\log \|\mathcal{O}\| +$	1	1.375
	$dist_{Dia}$	$\|S\| \log \|S\| + \sum_{o \in S} \|o.\psi\|))$		$\sqrt{3}$
$cost_{Sum}$	$dist_{MaxSum}$	$O(\|\mathcal{O}''\| \cdot (\log \|\mathcal{O}\|$	$O(\log \|q.\psi\|)$	1.375
	$dist_{Dia}$	$+\|\psi\|^2 \|S\|))$		$\sqrt{3}$

have $cost_{Sum}(G') \leq O(\log |q.\psi|) \cdot w(\tilde{G}) \leq O(\log |q.\psi|) \cdot w(G^*)$ and the proof is complete. □

Time Complexity. The time cost of CD-Appro is $O(|\mathcal{O}''| \cdot (\log |\mathcal{O}| + |\psi|^2 |S|))$ where $|\psi|^2 |S|$ is the time complexity of SumGreedy. □

5.2 Extension to the distance function $dist_{Dia}$

For $dist_{Dia}$, the algorithms in Sections 3 and 4 can still be applied except that we need to change Step 2, i.e., the candidate set construction step, as follows.

5.2.1 CD-Exact

We set $d_1 = d(o, q)$ and $d_2 = B$ for $dist_{Dia}$. The intuition of this change is that we do not want to miss any feasible candidate sets for $dist_{Dia}$. A candidate set wrt the given seed object o is defined as the set of relevant objects in the region $D(o, d_2)$. The other parts remain the same. We can observe that the time complexity remains the same since we only change the region $D(o, d_2)$.

5.2.2 CD-Appro

We set $d_1 = d(o, q)$ and $d_2 = B$ for $dist_{Dia}$. Then a candidate set with regard to a given seed object o is defined as the set of relevant objects in the region $D(o, d_2) \cap D(q, d_1)$. The other parts remain the same. Again, it is easy to observe that the time complexity for the algorithm remains the same as we only modify the values of d_1 and d_2.

Lemma 6. CD-Appro gives a distance ratio β of $\sqrt{3}$. □

Proof: Consider a candidate set S constructed from o. By Theorem 2 of [21], we know that $\max\{\max_{o \in S} d(o, q), \max_{o_1, o_2 \in S} d(o_1, o_2)\} \leq \sqrt{3} \max\{d_1, d_2\}$. Thus, we have $dist_{Dia}(S) \leq \sqrt{3} \cdot B$. Since the set G returned by CD-Appro (if any) corresponds to a subset of a candidate set S, we know $dist_{Dia}(G) \leq dist_{Dia}(S) \leq \sqrt{3} \cdot B$. □

5.3 Summaries

Table 3 shows the time complexities and the approximation ratios of CD-Appro under different settings of cost functions and distance functions.

6 Empirical Studies

6.1 Experimental Set-up

Datasets. We used three real datasets in our experiments, namely Yelp, Hotel and GN. Dataset Yelp was extracted from Yelp Academic Data Set[1] and was used in [13], where each POI (i.e., object) has a location and belongs to a set of categories (e.g., Dim Sum, Dry Cleaning, and Pubs). We use the set of categories as the keywords of an object. Each object is rated by a 5-star scale (with a 0.5 interval), and we converted it to the range $[1, 10]$.

1. https://www.yelp.com/dataset/challenge

TABLE 4
Datasets used in the experiments

	Yelp	Hotel	GN
# of objects	192,609	20,790	1,868,821
# of unique words	2,468	602	222,409
# of words	788,841	80,645	18,374,228

Datasets Hotel and GN were used in [4], [21], [3], [7], [8]. 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 and 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). Following [13], we generated the object weights in the range $[1, 10]$ following a normal distribution with mean 5 and standard deviation 1. Table 4 shows the statistics of the three datasets.

Query Generation. Let $\mathcal{O}$ be a dataset of objects. Given an integer k, we generated a query q with k query keywords similarly as [4], [21], [7], [8] did. Specifically, to generate $q.\lambda$, we randomly picked a location from the MBR of the objects in $\mathcal{O}$, and to generate $q.\psi$, we first ranked all the keywords that are associated with objects in $\mathcal{O}$ in descending order of their frequencies and then randomly picked k keywords in the percentile range of [10, 40]. In this way, each query keyword has a relatively high frequency.

Algorithms. We studied our CD-Exact and CD-Appro. For comparison, we have the following algorithms.

- **Combi-Exact.** It is a baseline algorithm which searches over all combinations of objects from those inverted lists corresponding to those query keywords in $q.\psi$ (note that all these combinations are feasible sets) and then finds the one which satisfies the distance threshold and has the smallest cost.
- **Cao-Appro.** It is MAXMAX-Appro2 proposed in [3] which is an approximation algorithm for the CoSKQ problem. Since the algorithm was not designed for finding a feasible set with the smallest cost (but a feasible set with the smallest distance), we adapt it as follows. Instead of finding a feasible set in each iteration and returning the one with the smallest distance as the solution, it returns the one with the smallest cost. The procedure within each iteration remains the same.
- **Long-Appro.** It is the approximation algorithm MaxSum-Appro proposed in [21] for the CoSKQ problem. We adapt it in a way similar to that of Cao-Appro to return the feasible set with the smallest cost among all processed object sets.

All experiments were conducted on a Linux platform with a 2.66GHz machine and 32GB RAM. The augmented IR-tree mentioned in Section 2.2 is used in the algorithms and the index structure is memory resident.

6.2 Experimental Result

Following the existing studies [4], [21], [3], [7], [8], we measured the running time and the approximation ratios, i.e., the cost ratio and the distance ratio, (for approximation algorithms only). For each experimental setting, we generated 50 queries and reported the average results.

6.2.1 Setting the default distance threshold B

We first find the default value of B. It is important to find a suitable value of B, since setting an extremely small value of B will give no result for the query, while setting B to a large value the solution would not be desirable to a user since he/she needs to traverse a long distance. So, we want to find a value of B such that a result could be found and the value is as small as possible. Specifically, we estimate the value of B as follows. We first run the approximation algorithm [21] for the CoSKQ problem, and use the distance of the solution as a lower bound on the distance threshold, denoted by $dist_{LB}$. We then set $B = dist_{LB} \times n$, where n is a user parameter, and vary n from 1.0 to 1.5.

The results could be found in Appendix B, available online, due to page limit. According to the results, the average costs of the solutions decrease when B increases. Still, the rate of decrease is very small when $n > 1.1$. Thus, we set the default value of B to 1.1 times $dist_{LB}$.

6.2.2 Effect of $|q.\psi|$

Following the existing studies [4], [21], [7], [8], we vary the number of query keywords from $\{3, 6, 9, 12, 15\}$.

Dataset Yelp. The results with $cost_{Max}$ and $dist_{MaxSum}$ are presented in Figure 4. According to Figure 4(a), the running times of the exact algorithms increase when $|q.\psi|$ increases. Our exact algorithm CD-Exact runs consistently faster than the Combi-Exact. This is because that the pruning strategies in CD-Exact can reduce the search space effectively.

According to Figure 4(b), our CD-Appro runs faster than Cao-Appro and Long-Appro. It is because CD-Appro has cost-related pruning techniques, which can help to reduce the number of iterations in the algorithm. Besides, CD-Appro can achieve the cost ratio α very close to 1 (and smaller than 1 in some cases), while that of Cao-Appro and Long-Appro are much larger, especially when $|q.\psi|$ is large (e.g., when $|q.\psi| \geq 9$, the cost ratios of Cao-Appro and Long-Appro are at least 1.4). Note that it is possible that $\alpha < 1$ because the relaxed distance threshold provide the flexibility to find a solution with cost lower than the optimal solution. The distance ratio β of CD-Appro is close to 1, and that of the Cao-Appro and Long-Appro are slightly smaller. This is probably because both Cao-Appro and Long-Appro were originally designed for the CoSKQ problem, which target at minimizing the distance function. In fact, while the distance constraint may be violated in CD-Appro, in 204 out of 250 cases (50 queries × 5 query sizes), the distance ratio is less than 1.1 and in 167 out of 250 cases, the distance ratio is satisfied, i.e., the distance constraint is satisfied. In case the user insists on satisfying the distance constraint, the CD-Exact algorithm can be adopted. Besides, we have CD-Appro runs an average of 36% faster than CD-Exact, since CD-Appro adopts the $MaxGreedy$ which finds a feasible set greedily.

The results with $cost_{Sum}$ and $dist_{MaxSum}$ on the dataset Yelp are presented in Figure 5, where the results for Combi-Exact with $|q.\psi| \geq 9$ are not shown because it runs for more than 10 hours. (This applies to most of the following results.) According to Figure 5(a), our CD-Exact runs faster than Combi-Exact by orders of magnitude. According to Figure 5(b), our CD-Appro runs faster than Cao-Appro and Long-Appro. The cost ratio α of CD-Appro is very close to 1, while that of Cao-Appro and Long-Appro are at least 1.39 and 1.58, respectively. All approximation algorithms can achieve the distance ratios close to 1, though that of CD-Appro is slightly larger than the competitors. It is noteworthy that CD-Appro runs faster than CD-Exact by an order of magnitude, due to the fact that CD-Appro uses a greedy procedure instead of an expensive enumeration.

The results with $cost_{Max}$ ($cost_{Sum}$) and $dist_{Dia}$ are presented in Figure 6 (Figure 7), which are similar to those for $dist_{MaxSum}$, i.e., our exact algorithm CD-Exact runs faster than Combi-Exact, our approximation algorithm CD-Appro runs quite fast, achieves cost ratios α very close to 1 (while that of the competitors are not) and distance ratios β smaller than 1 in practice.

Datasets Hotel and GN. Due to page limit, the results are presented in Appendix C, available online.

6.2.3 Effect of B

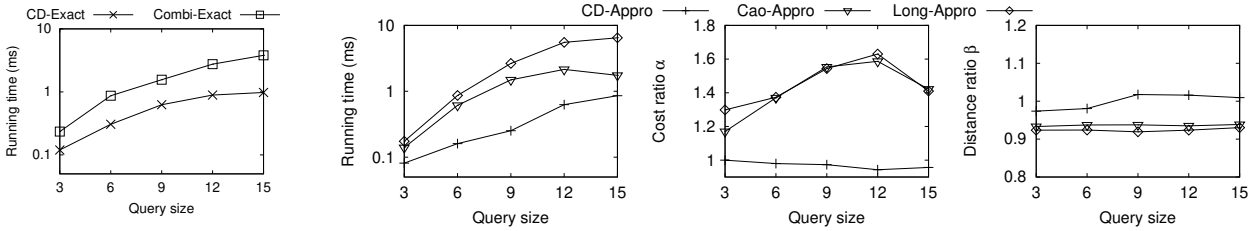
We also studied the effect of distance threshold B by setting $B = dist_{LB} \times n$, where $dist_{LB}$ is the distance cost of the solution found by the approximation algorithm [21] for the CoSKQ problem. We vary n from $\{1.0, 1.05, 1.1, 1.15, 1.20\}$. The default value of $|q.\psi| = 6$. The results on running times and approximation ratios can be found in Appendix D, available online.

Figure 8 reports the average candidate set size, which shows that, for both $dist_{MaxSum}$ and $dist_{Dia}$, the set size increases when B increases, and the set size of CD-Appro is always smaller than that of CD-Exact, which is due to the fact that the candidate set region of CD-Appro is much smaller. Moreover, the candidate set size of $dist_{Dia}$ is consistently larger than that of $dist_{MaxSum}$, which is also related to the size of the candidate set region, that $dist_{Dia}$ has a larger region.

6.2.4 Scalability Test

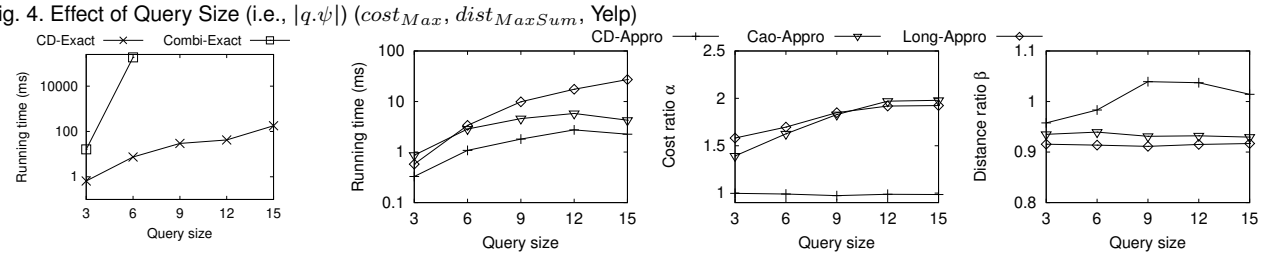
Following the existing studies [4], [21], [3], [7], [8], we generated 5 synthetic datasets for the experiments of scalability test, in which the numbers of objects used are $\{2M, 4M, 6M, 8M, 10M\}$ where M represents million. The datasets were generated as follows. We generated a synthetic dataset by augmenting the GN dataset. Each time, we created a new object o with $o.\lambda$ set to be a random location from GN dataset by following the distribution and $o.\psi$ set to be a random document from GN and then add it into the GN dataset. We followed [21], [7] and used the default setting of $|q.\psi| = 6$.

The scalability test results with $cost_{Max}$ and $dist_{MaxSum}$ are presented in Figure 9. According to Figure 9(a), the running times of the algorithms increase when the number of objects in the datasets increases, and CD-Exact and Combi-Exact have similar running times. According to Figure 9(b), our CD-Appro is scalable to large datasets, e.g., they ran



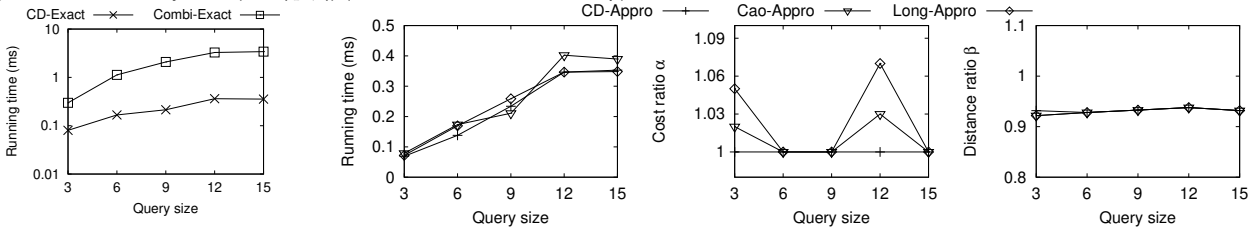
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 4. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Max}$, $dist_{MaxSum}$, Yelp)



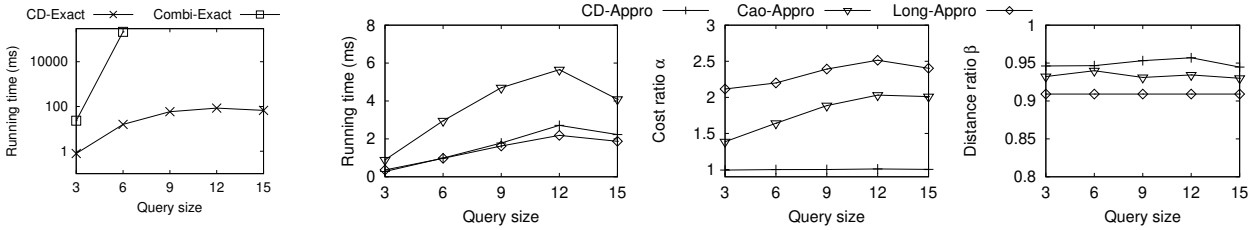
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 5. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Sum}$, $dist_{MaxSum}$, Yelp)



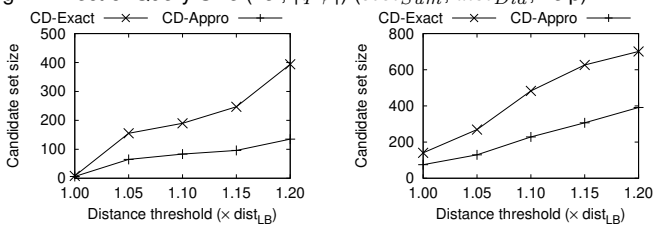
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 6. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Max}$, $dist_{Dia}$, Yelp)



(a) Exact Algorithms (b) Approximation Algorithms

Fig. 7. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Sum}$, $dist_{Dia}$, Yelp)



(a) $dist_{MaxSum}$ (b) $dist_{Dia}$

Fig. 8. Effect of candidate set size on B (Yelp)

within 1s on a dataset with 10M objects. The cost ratio α of CD-Appro is always smaller than 1, and is much smaller than that of Cao-Appro and Long-Appro. The distance ratio β of CD-Appro is larger than Cao-Appro and Long-Appro. It is because when the number objects in the datasets increases, the number of objects in the candidate sets increases which result in a feasible set with larger cost is found.

The results on other settings are presented in Appendix E, available online.

6.2.5 Case Studies

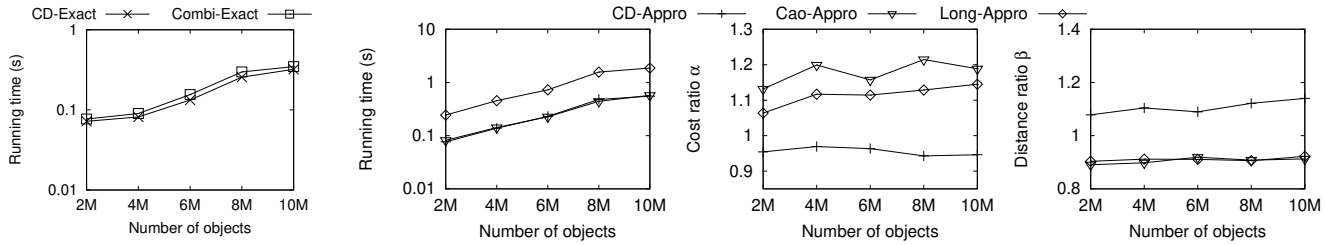
To evaluate the quality of the results, we compare the results of traditional CoSKQ (i.e., MaxSum-CoSKQ and Dia-CoSKQ) with our CD-CoSKQ. Table 5 shows the result of optimal solution. According to Table 5, our CD-CoSKQ is able to find a solution that has a much lower cost, with a slightly larger distance compared to those that minimize distances. In particular, with less than 10% increases in distances, we can reduce the cost of the solution by more than a half, for both $cost_{Max}$ and $cost_{Sum}$.

TABLE 5
Case studies on dataset Yelp

		CoSKQ		CD-CoSKQ	
		$dist$	$cost$	$dist$	$cost$
$dist_{MaxSum}$	$cost_{Max}$	30.404	7.500	32.093	2.680
	$cost_{Sum}$		28.520	31.962	12.760
$dist_{Dia}$	$cost_{Max}$	30.342	7.500	31.887	2.620
	$cost_{Sum}$		29.240	31.953	12.400

7 Related Works

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* [16], [6], [31], [38], [35] 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* [11], [23], [20], [24], [25], [12],



(a) Exact Algorithms (b) Approximation Algorithms

Fig. 9. Scalability Test ($cost_{Max}$, $dist_{MaxSum}$)

[32] 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 [11], [23], [20], road network databases [24], trajectory databases [25], [12], and moving object databases [32]. Usually, the methods for this kind of queries adopt an index structure called the *IR-tree* [11], [30] 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, there exist works [4], [21], [3], [7], [8], [26] studying the *collective spatial keyword queries* (CoSKQ). Cao et al. [4], [3] studied the CoSKQ problem and proposed exact and approximation algorithms for several distance functions, e.g., $dist_{MaxSum}$. Besides, they studied two variations of CoSKQ, namely the top-k CoSKQ and the weighted CoSKQ, in [3]. Long et al. [21] proposed exact and approximation algorithms with improved performance for the CoSKQ problem with the distance function $dist_{MaxSum}$ and also that with a new distance function $dist_{Dia}$. Chan et al. [8] studied a generalized distance function, which unifies some of the previous distance functions (e.g., $dist_{MaxSum}$ and $dist_{Dia}$). They also proposed a generalized framework for the exact and approximate solutions. Xu et al. [33] studied the moving CoSKQ problem and proposed both exact and approximation algorithms for the problem. Song et al. [26] studied CoSKQ problem on activity trajectories and proposed an index structure and a search algorithm for the problem.

There are existing studies which take into account not only the geospatial information but also some attribute information (such as popularity and expenses of objects) for defining spatial keyword queries [7], [39]. Chan et al. [7] define a cost-aware collective spatial keyword query to find a set of objects such that the objects cover all the query keywords and the set has the smallest product of its cost and distance. As mentioned in Section 1, this query does not provide users the flexibility to express their finer grained preferences. Zhao et al. [39] studied the popularity-aware CoSKQ on road networks, which define a query to find a region consisting of POIs such that the POIs cover the query keywords, the diameter of the region is at most some threshold, the maximum traveling distance from a POI to a query location is at most another threshold, and the set has the sum of the popularities of the POIs the largest. This study is similar to ours in that it uses some form of distance based on geospatial information as constraints and the popularity as the objective, but they differ in that this study is based on road networks while ours on Euclidean space, this study considers only one cost function while ours consider multiple cost functions and distance functions, and the approximation algorithms in this study provide no guarantees while our approximation algorithms provide provable guarantees.

Another query that is similar to the CoSKQ problem is the *m-Closest Keywords (mCK) Query* [36], [37], [18] which takes a set of m keywords as input and finds a set of objects with the minimum *diameter* that cover the m keywords specified in the query. Guo et al. [18] gave the state-of-the-art exact and approximation algorithms. There are some variants of the mCK query, including the *SK-COVER* [9], [10] and the *BKC query* [13]. 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 (without a query location). In contrast, the CD-CoSKQ problem studied in this paper takes both a set of keywords and a spatial location as inputs.

There are also some studies [17], [27], [39] on spatial keyword queries which find an object set in the road network, e.g., [5], [14] find a *region* as a solution and [2], [34], [19], [29], [40] find a *route* as a solution.

8 Conclusion

In this paper, we proposed a new type of query called CD-CoSKQ, which aims to find an object set with the smallest cost subject to a distance constraint. We proved that the CD-CoSKQ problem is NP-hard to approximate with any constant factor. We developed one exact algorithm and one approximation algorithm with provable guarantees for the problem. Extensive experiments were conducted which verified our theoretical findings.

There are several interesting future research directions. One direction is to explore other distance functions, e.g., $dist_{Sum}$, $dist_{MinMax}$, for the distance constraint. Another direction is to extend CD-CoSKQ problem to other distance metrics such as road network. Besides, it would be interesting to consider multiple attributes in the objects and integrate them into one single cost function.

Acknowledgments: We thank anonymous reviewers for their helpful comments. This research is supported by the Nanyang Technological University Start-Up Grant from the College of Engineering under Grant M4082302 and by

the Ministry of Education, Singapore, under its Academic Research Fund Tier 1 (RG20/19 (S)). The research of the HKUST side is supported by IRS17EG25.

REFERENCES

[1] A. Aggarwal, A. Deshpande, and R. Kannan. Adaptive sampling for k-means clustering. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, pages 15–28. Springer, 2009.

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

[3] 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.

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

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

[6] 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.

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

[8] H. K.-H. Chan, C. Long, and R. C.-W. Wong. On generalizing collective spatial keyword queries. *TKDE*, 30(9):1712–1726, 2018.

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

[10] D.-W. Choi, J. Pei, and X. Lin. On spatial keyword covering. *Knowledge and Information Systems*, pages 1–36, 2020.

[11] 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.

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

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

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

[15] D. Feldman, A. Fiat, M. Sharir, and D. Segev. Bi-criteria linear-time approximations for generalized k-mean/median/center. In *Proceedings of the twenty-third annual symposium on Computational geometry*, pages 19–26, 2007.

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

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

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

[19] W. Li, J. Cao, J. Guan, M. L. Yiu, and S. Zhou. Efficient retrieval of bounded-cost informative routes. *TKDE*, 29(10):2182–2196, 2017.

[20] 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.

[21] 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.

[22] K. Makarychev, Y. Makarychev, M. Sviridenko, and J. Ward. A bi-criteria approximation algorithm for k means. *arXiv preprint arXiv:1507.04227*, 2015.

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

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

[25] 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.

[26] X. Song, J. Xu, R. Zhou, C. Liu, K. Zheng, P. Zhao, and N. Falkner. Collective spatial keyword search on activity trajectories. *GeoInformatica*, 24(1):61–84, 2020.

[27] 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.

[28] V. V. Vazirani. *Approximation algorithms*. Springer, 2013.

[29] Y.-T. Wen, J. Yeo, W.-C. Peng, and S.-W. Hwang. Efficient keyword-aware representative travel route recommendation. *TKDE*, 29(8):1639–1652, 2017.

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

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

[32] 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.

[33] H. Xu, Y. Gu, Y. Sun, J. Qi, G. Yu, and R. Zhang. Efficient processing of moving collective spatial keyword queries. *VLDBJ*, 29(4):841–865, 2020.

[34] 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.

[35] 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.

[36] 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.

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

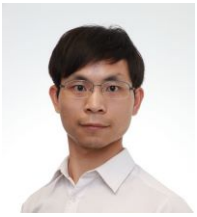
[38] D. Zhang, K.-L. Tan, and A. K. H. Tung. Scalable top-k spatial keyword search. EDBT/ICDT, pages 359–370. ACM, 2013.

[39] S. Zhao, X. Cheng, S. Su, and K. Shuang. Popularity-aware collective keyword queries in road networks. *Geoinformatica*, 21(3):485–518, 2017.

[40] S. Zhao, L. Zhao, S. Su, X. Cheng, and L. Xiong. Group-based keyword-aware route querying in road networks. *Information Sciences*, 450:343–360, 2018.

Harry Kai-Ho Chan is currently a Post-Doctoral Researcher at Department of People and Technology, Roskilde University, Denmark. He received the BEng, MPhil and PhD degrees in computer science and engineering from the Hong Kong University of Science and Technology (HKUST) in 2013, 2015 and 2019, respectively. His research interests include database, data mining and indoor location-based services.

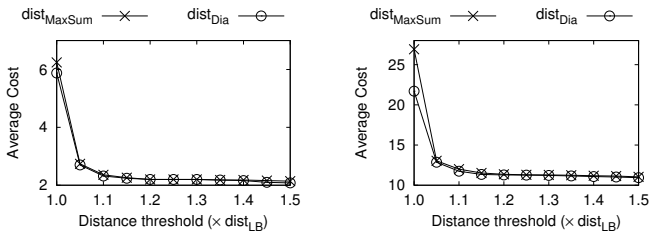
Shengxin Liu is currently an Assistant Professor at School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He received the Ph.D. degree from the Department of Computer Science, the City University of Hong Kong, China, and worked as a Post-Doctoral Research Fellow with the School of Physical and Mathematical Sciences, Nanyang Technological University, Singapore. His research interests include computational social choice, data structures and algorithms, and algorithmic databases. He is the recipient of Outstanding Student Paper Award at AAAI 2020 and Best Paper Award at FAW 2020.



Cheng Long (S'11-M'15) is currently an Assistant Professor at the School of Computer Science and Engineering, Nanyang Technological University. He received his PhD degree from the Hong Kong University of Science and Technology, Hong Kong, in 2015, and his BEng degree from South China University of Technology, China, in 2010. His research interests are broadly in data management, data mining and big data analytics.

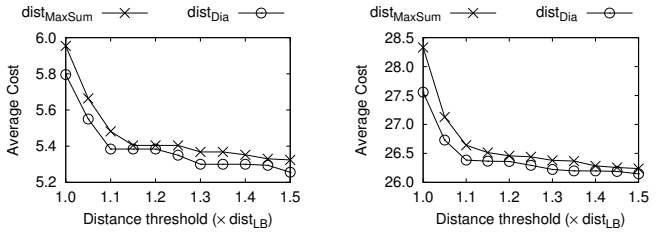
Raymond Chi-Wing Wong received the BSc, MPhil and PhD degrees in computer science and engineering from the Chinese University of Hong Kong (CUHK) in 2002, 2004, and 2008, respectively. He is a professor of the Department of Computer Science and Engineering, the Hong Kong University of Science and Technology. His research interests include database and data mining.

Cost-Aware and Distance-Constrained Collective Spatial Keyword Query (Appendix)



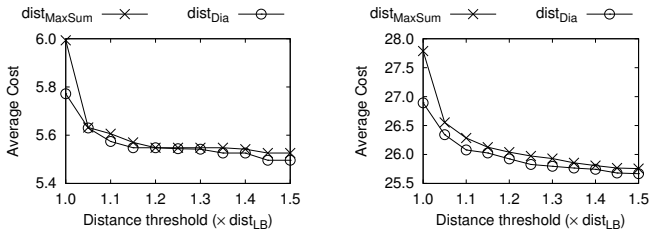
(a) $cost_{Max}$ (b) $cost_{Sum}$

Fig. 10. Effect of average cost on B (Yelp)



(a) $cost_{Max}$ (b) $cost_{Sum}$

Fig. 11. Effect of average cost on B (Hotel)



(a) $cost_{Max}$ (b) $cost_{Sum}$

Fig. 12. Effect of average cost on B (GN)

APPENDIX A
PROOF OF THE NP-HARDNESS OF APPROXIMATION

Proof: We prove this theorem by a reduction from the collective spatial keyword query (CoSKQ) problem [21]. Given a query q with a location $q.\lambda$ and a set of keywords $q.\psi$, the CoSKQ problem is to find a set of objects G such that (1) they cover all the query keywords and (2) the distance of G, $dist(G)$, is minimized. The decision problem of the CoSKQ problem is that given a problem instance of CoSKQ and a value C, it checks whether there exists a set of objects G such that G covers $q.\psi$ and $dist(G) < C$. It has been shown in [21] that the CoSKQ problem with both the $dist_{Dia}(G)$ function and the $dist_{MaxSum}(G)$ function is NP-hard.

We prove by contradiction. Suppose that we have a polynomial-time c-approximation algorithm $\mathcal{A}$ for the CD-CoSKQ problem with $c \geq 1$. In other words, in the case that the problem instance of CD-CoSKQ has feasible solutions, $\mathcal{A}$ would return a feasible solution with its cost at most c times the cost of the optimal solution; and it returns an empty set otherwise. It follows that this algorithm could be used solve the decision problem of the CoSKQ problem as follows.

Given the decision problem of a CoSKQ instance, we run $\mathcal{A}$ with the query location and query keywords the same as those of the CoSKQ problem and the distance threshold B as C. Then, if $\mathcal{A}$ returns a non-empty solution, we conclude that the answer to the decision problem is yes; and otherwise, no. Thus, this leads to a contradiction which finishes the proof. □

APPENDIX B
SETTING THE DEFAULT DISTANCE THRESHOLD B

The results for the dataset Yelp are shown in Figure 10. According to the results, the average costs of the solutions decrease when the distance threshold B increases. Still, the rate of decrease is very small when $n > 1.1$. Thus, we set the default value of B to 1.1 times $dist_{LB}$.

The results for the dataset Hotel and GN with different distance threshold give similar clues and are shown in Figure 11 and Figure 12, respectively.

APPENDIX C
EFFECT OF QUERY SIZE

Dataset Hotel. The results with $cost_{Max}$ and $dist_{MaxSum}$ are presented in Figure 13. According to Figure 13(a), the running times of the algorithm increase when $|q.\psi|$ increase, and our CD-Exact runs faster than Combi-Exact. According to Figure 13(b), our CD-Appro runs faster than Cao-Appro and Long-Appro, and it can always achieve cost ratio α smaller than 1. Besides, the distance ratio β of CD-Appro is slightly larger than Cao-Appro and Long-Appro, but is close to 1.

The results with $cost_{Sum}$ and $dist_{MaxSum}$ are presented in Figure 14. According to Figure 14(a), our CD-Exact runs faster than Combi-Exact, and their difference increases with the query size. According to Figure 14(b), the approximation algorithms have similar running times. CD-Appro achieve better cost ratios than Cao-Appro and Long-Appro consistently, while CD-Appro has the distance ratios close to 1.

The results with $cost_{Max}$ and $dist_{Dia}$ for dataset Hotel are similar and are presented in Figure 15. The results with $cost_{Sum}$ and $dist_{Dia}$ for dataset Hotel are presented in Figure 16.

Dataset GN. The results with $cost_{Sum}$ and $dist_{MaxSum}$ are presented in Figure 18. According to Figure 18(a), our CD-Exact runs faster than Combi-Exact, especially when query size is large. According to Figure 18(b), CD-Appro and Cao-Appro have similar running times, while Long-Appro is much slower. Besides, CD-Appro always achieve the cost ratio close to 1, and outperform Cao-Appro and

Long-Appro. All of them have distance ratios close to 0.9 and smaller than 1.

The results with $cost_{Max}$ and $dist_{Dia}$ for dataset GN are presented in Figure 19. The results with $cost_{Sum}$ and $dist_{Dia}$ for dataset GN are presented in Figure 20.

APPENDIX D
EFFECT OF B

We set the distance threshold $B = dist_{LB} \times n$, where $dist_{LB}$ is the distance cost of the solution found by the approximation algorithm [21] for the CoSKQ problem. We vary n from $\{1.0, 1.05, 1.1, 1.15, 1.20\}$. The default value of $|q.\psi| = 6$.

The results with $cost_{Max}$ and $dist_{MaxSum}$ on the dataset Yelp are shown in Figure 21. According to Figure 21(a), the running times of both CD-Exact and Combi-Exact do not change much when B increases, and CD-Exact is much faster than Combi-Exact. It is probably because when B increases, the number of relevant objects increases, but at the same time it would be easier to find the feasible set with minimum cost in an iteration since the budget is relaxed.

According to Figure 21(b), the running times of the approximation algorithms do not change much when B increases, and both CD-Appro runs much faster than Cao-Appro and Long-Appro. For CD-Appro, it is probably because when B increases, the number of objects processed increases with the number of key objects. But on the other hand, a better solution could possibly be found within each iteration, and reducing the total number of iterations needed. Thus, the overall running times remain similar. Besides, the cost ratios α of the approximation algorithms increase when B increases, while that of CD-Appro remains close to 1. The reason is that a larger B could allow a smaller cost in the optimal solution, but Cao-Appro and Long-Appro cannot fully utilize this advantage, while our CD-Appro is able to explore possible better solutions. Besides, the distance ratio β of the approximation algorithms decrease when B increases. This is simply because B is the denominator in calculating the distance ratios.

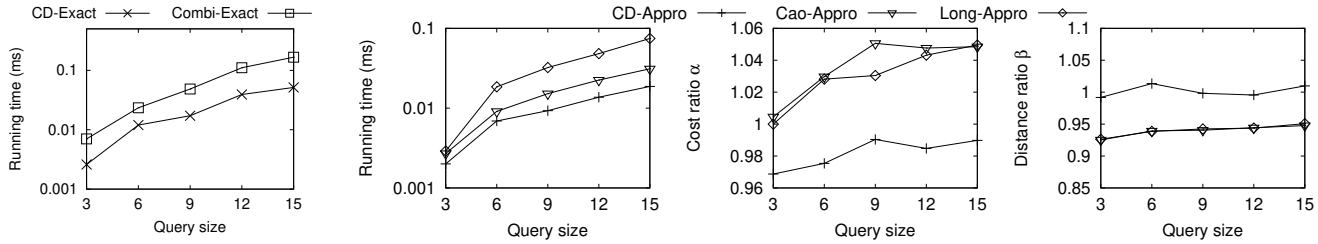
The results with $cost_{Sum}$ and $dist_{MaxSum}$ on the dataset Yelp are shown in Figure 22. According to Figure 22(a), the running times of both CD-Exact and Combi-Exact increase when B increases, and CD-Exact is much faster than Combi-Exact. According to Figure 22(b), CD-Appro runs faster than Cao-Appro and Long-Appro, and only increase slightly when B increases.

The results on $dist_{Dia}$ provide similar clues and are presented in Figure 23 and Figure 24.

APPENDIX E
SCALABILITY TEST

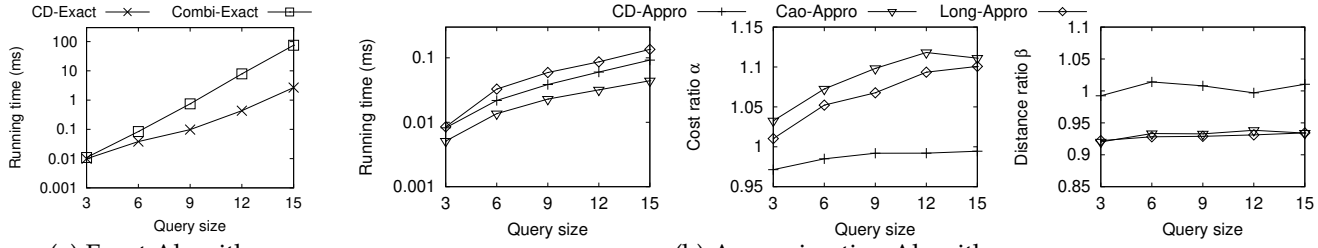
The scalability test results with $cost_{Sum}$ and $dist_{MaxSum}$ are presented in Figure 25. According to Figure 25(a), our CD-Exact is scalable wrt to the number of objects in the datasets, e.g., it ran within 10s on a dataset with 10M objects. Besides, according to Figure 25(b), our CD-Appro is scalable to large datasets, e.g., they ran within 2s on a dataset with 10M objects. The cost ratio α of CD-Appro is always smaller than 1, while its distance ratio β is slightly larger than 1.

The results with $cost_{Max}$ with $dist_{Dia}$ are similar and are presented in Figure 26. The results with $cost_{Sum}$ with $dist_{Dia}$ are presented in Figure 27.



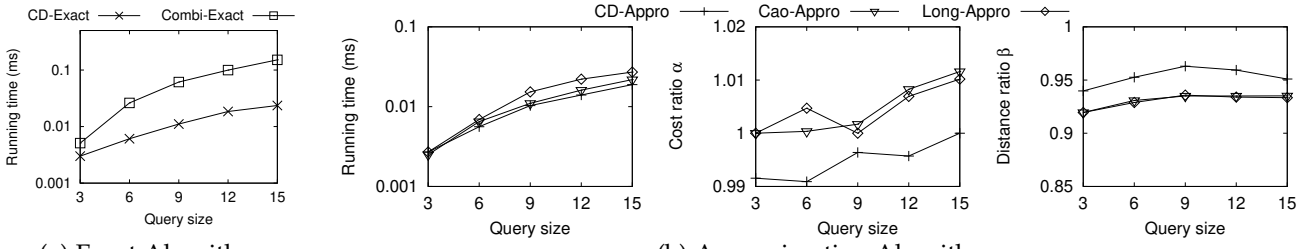
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 13. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Max}$, $dist_{MaxSum}$, Hotel)



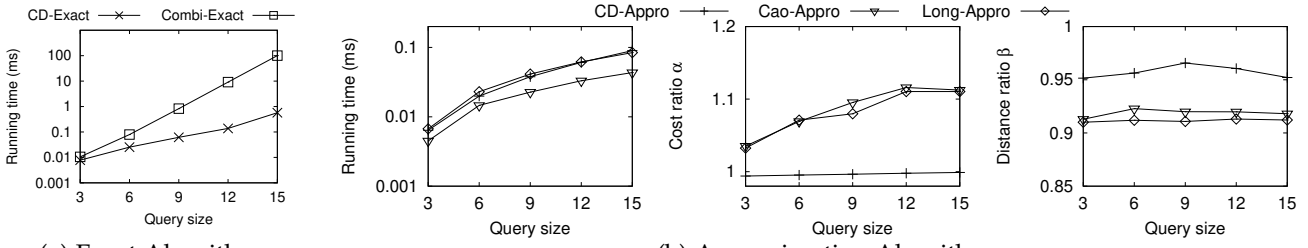
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 14. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Sum}$, $dist_{MaxSum}$, Hotel)



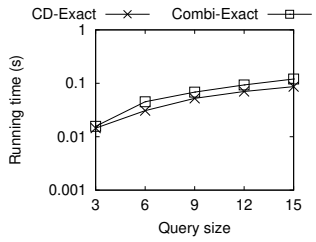
(a) Exact Algorithms (b) Approximation Algorithms

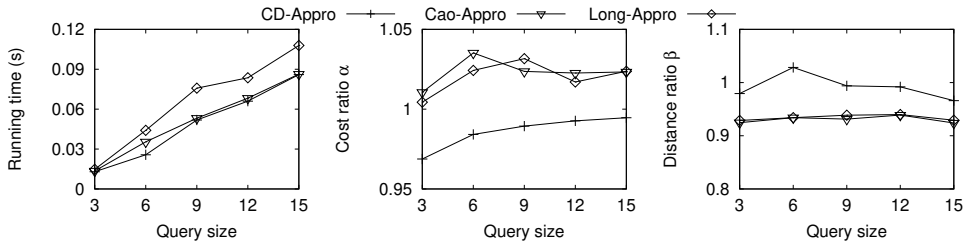
Fig. 15. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Max}$, $dist_{Dia}$, Hotel)



(a) Exact Algorithms (b) Approximation Algorithms

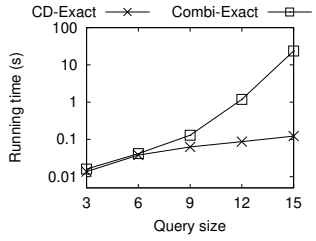
Fig. 16. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Sum}$, $dist_{Dia}$, Hotel)

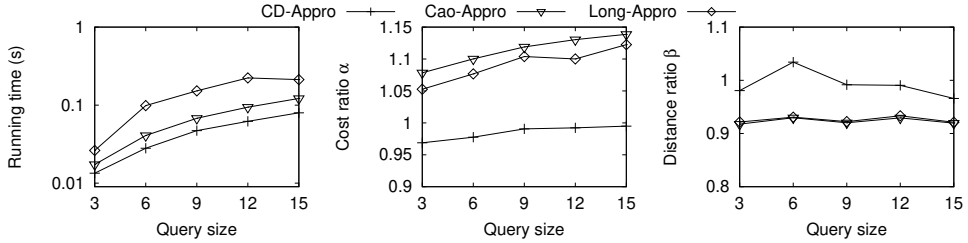




(a) Exact Algorithms (b) Approximation Algorithms

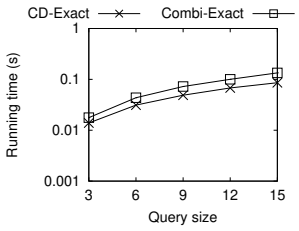
Fig. 17. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Max}$, $dist_{MaxSum}$, GN)

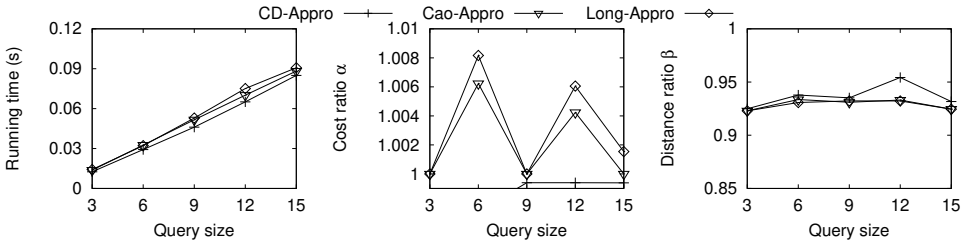




(a) Exact Algorithms (b) Approximation Algorithms

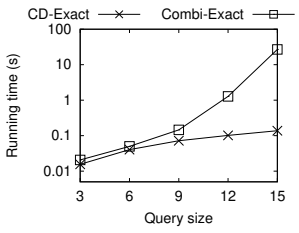
Fig. 18. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Sum}$, $dist_{MaxSum}$, GN)

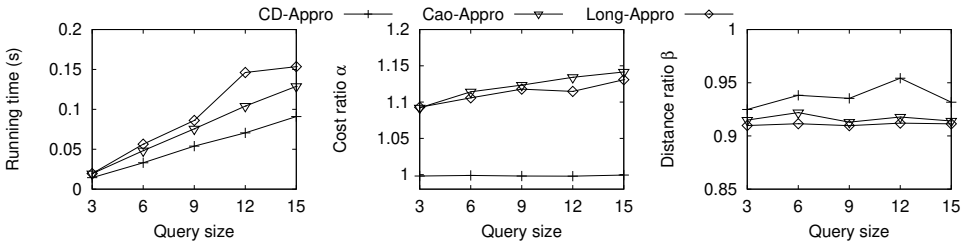




(a) Exact Algorithms (b) Approximation Algorithms

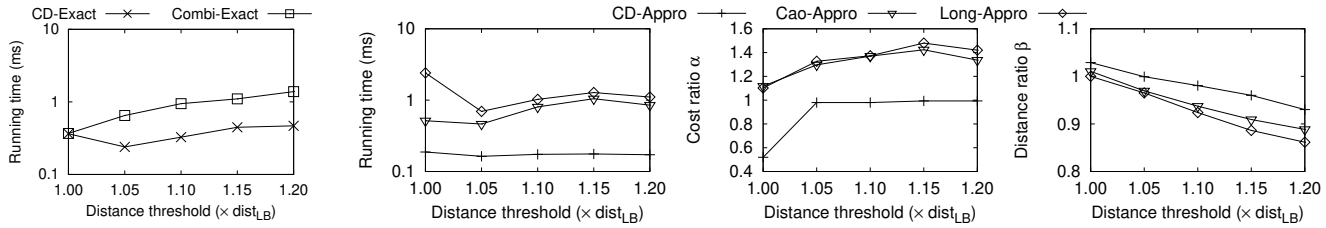
Fig. 19. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Max}$, $dist_{Dia}$, GN)





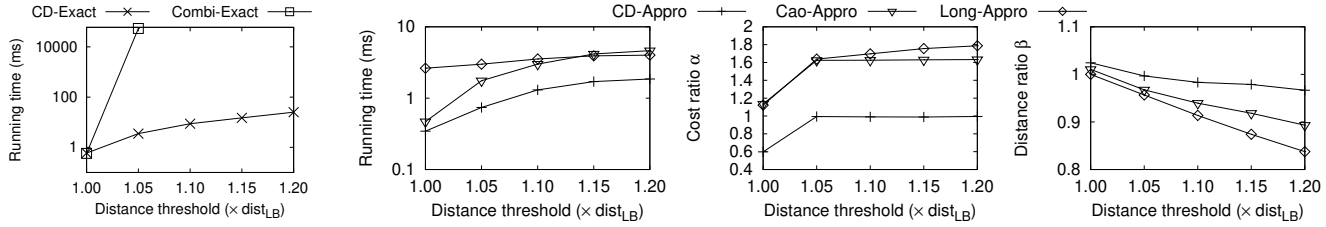
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 20. Effect of Query Size (i.e., $|q.\psi|$) ($cost_{Sum}$, $dist_{Dia}$, GN)



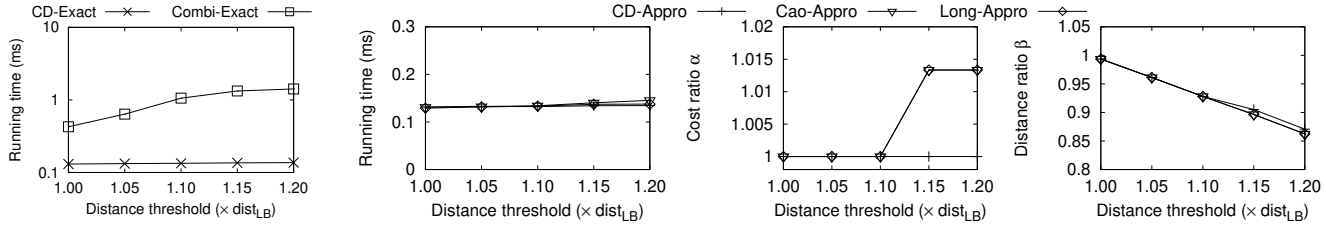
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 21. Effect of B ($cost_{Max}$, $dist_{MaxSum}$, Yelp)



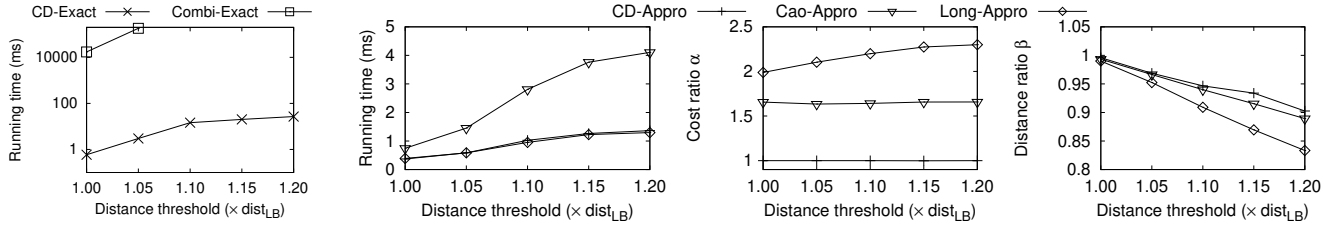
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 22. Effect of B ($cost_{Sum}$, $dist_{MaxSum}$, Yelp)



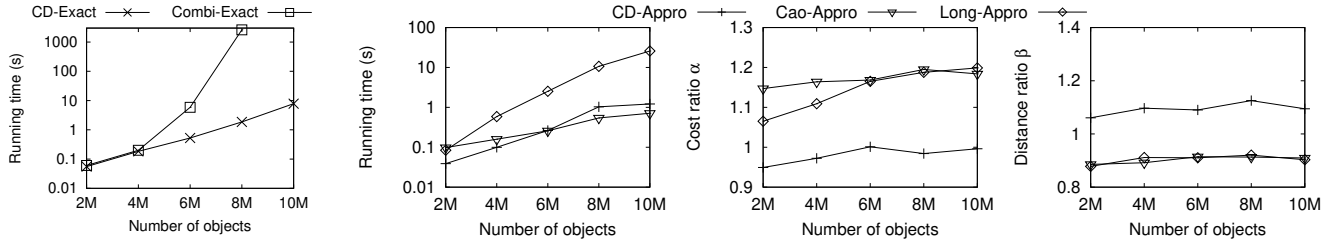
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 23. Effect of B ($cost_{Max}$, $dist_{Dia}$, Yelp)



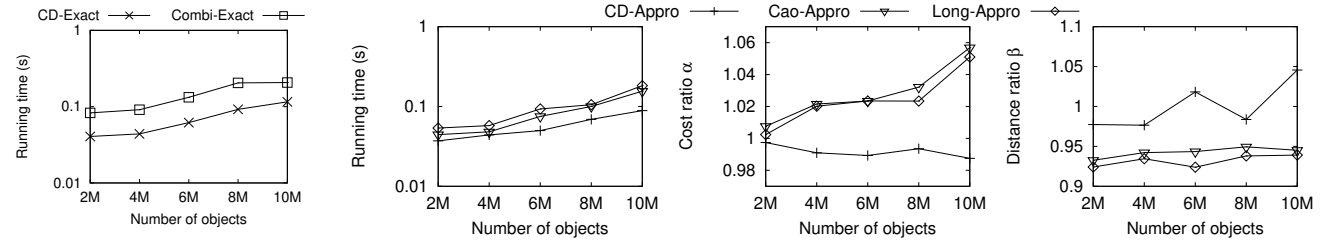
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 24. Effect of B ($cost_{Sum}$, $dist_{Dia}$, Yelp)



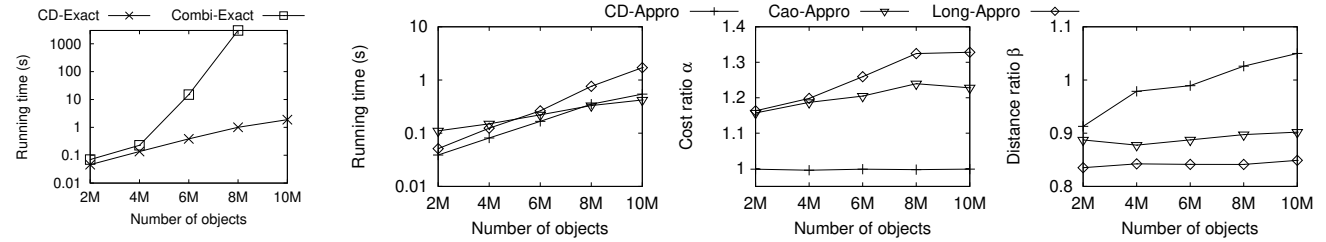
(a) Exact Algorithms (b) Approximation Algorithms

Fig. 25. Scalability Test ($cost_{Sum}$, $dist_{MaxSum}$)



(a) Exact Algorithms (b) Approximation Algorithms

Fig. 26. Scalability Test ($cost_{Max}$, $dist_{Dia}$)



(a) Exact Algorithms (b) Approximation Algorithms

Fig. 27. Scalability Test ($cost_{Sum}$, $dist_{Dia}$)