# CSKQS: A Query System for Collective Spatial Keyword Queries

Harry Kai-Ho Chan
*Information School, University of Sheffield, United Kingdom*
h.k.chan@sheffield.ac.uk

***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 often numerical attributes (e.g., expenses and users' ratings for points of interest). One popular spatial keyword queries on geo-textual data is the Collective Spatial Keyword Query (CoSKQ), which 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 criterion. In this work, we demonstrate a Collective Spatial Keyword Queries System (CSKQS) that supports both the conventional CoSKQ, and a variant called Cost-Aware and Distance-Constrained CoSKQ. CSKQS adopts the client/server system architecture, enabling users to access it through a web browser. The client side interface allows users to specify their queries and view the results, while the server side handles query processing and stores the data. CSKQS also offers an interactive map visualization, providing an intuitive view of the spatial relationship between the result object set and query location.**

## I. Introduction

Nowadays, geo-textual data which refers to data with both spatial and textual information is ubiquitous. Examples of geo-textual data include 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). Many of these geo-textual objects also come with additional numerical attributes. For example, a restaurant is usually associated with some expense attribute (e.g., in Yelp, this information is shown by the number of "$" symbols).

Many types of spatial keyword queries have been proposed on geo-textual data. Among them, one prominent type is to search a set of (geo-textual) objects wrt a query consisting of a query location (e.g., the user's current location) and some query keywords expressing the targets that the user wants to search such that the objects have their textual information *matching* the query keywords and their locations close to the query location. For example, a tourist visiting a city wants to find several POIs for sight-seeing, shopping and dining, and all these POIs are close to the hotel. In this case, the tourist can set the hotel as the query location, and the query keywords to be "attraction", "shopping", and "restaurant".

This query type was captured by the *Collective Spatial Keyword Query* (CoSKQ) [2], [9], [1], [5], which 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. We call an object set $G$ a *feasible set* if its covers all query keywords. For a given query, there are usually many feasible sets - 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? Existing studies define the criterion either based on solely the geospatial aspects of the objects [1], [2], [5], [9], or by a variant that considers both the geospatial aspects and attribute aspects [3], [4].

In this work, we demonstrate a Collective Spatial Keyword Queries System (CSKQS) that supports the two types of CoSKQs: **(1)** CSKQS supports the conventional CoSKQ [5], which is to find the feasible set, i.e., the object set covers all query keywords $q.\psi$, which has the smallest *distance* wrt the query location $q.\lambda$, 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; **(2)** CSKQS also supports the *Cost-Aware and Distance-Constrained CoSKQ* (CD-CoSKQ) [3], which aims to find an object set with the smallest *cost* subject to a budget constraint $q.B$ on the *distance*. The cost is based on the attribute aspect of the objects and the distance on the geospatial aspect. CD-CoSKQ provides users a finer grained interface to express their preferences on both the geospatial aspect and the attribute aspect of the geo-textual objects.

Many systems have been developed to support different types of spatial keyword queries. For example, IndoorViz [7] addresses the indoor spatial keyword queries, and TASKS [10] supports the error-tolerant spatial keyword queries on road network. These systems, however, only support queries that return a single spatial object per result and are therefore not applicable to CoSKQ. ExampleSearcher [11] answers the keyword searches and spatial examplar queries, which return object sets as results. It uses an example-based search, and the output is based on the similarity with the example. Thus, it can not be used to capture the distance function employed in CoSKQ or the cost function utilized in CD-CoSKQ.

To the best of our knowledge, CSKQS is the first system designed to support collective spatial keyword queries. The main features of CSKQS are summarized as follows. First, it provides a user interface allowing users to query, set parameters and view results. Second, it adopts a client/server architecture, enabling users to access the system from a browser. Third, it offers an interactive map visualization to show the spatial relationship between the result object set and query location in an intuitive way.

The rest of this demonstration proposal is organized as follows. Section II introduces the problem definition of CoSKQs. Section III presents the architecture and design of CSKQS. Finally, Section IV shows the user interface and details the demonstration scenarios.

## II. Problem Definition

In this section we formally introduce the problem definition of CoSKQ and CD-CoSKQ.

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. Given two objects $o$ and $o'$, we denote by $d(o, o')$ the Euclidean distance between $o.\lambda$ and $o'.\lambda$.

The conventional CoSKQ is defined as follows.

*Problem 1 (CoSKQ [5]):* Given a query $q$ which consists of a query location $q.\lambda$ and a set of query keywords $q.\psi$, the CoSKQ problem is to find a set $G \subseteq \mathcal{O}$ of objects such that (1) $G$ covers $q.\psi$ (i.e., $G$ is a feasible set), and (2) the distance of $G$ wrt $q$, denoted by $dist(G, q)$, is minimized. □

**Distance Functions.** Given a query $q$ and an object set $G$, we consider the following six distance functions in this paper.

$$\begin{aligned}
dist_{MaxSum}(G,q) &= \max_{o\in G} d(o,q) + \max_{o_1,o_2\in G} d(o_1,o_2)\\
dist_{Dia}(G,q) &= \max_{o_1,o_2\in G\cup\{q\}} d(o_1,o_2)\\
dist_{Sum}(G,q) &= \sum_{o\in G} d(o,q)\\
dist_{SumMax}(G,q) &= \sum_{o\in G} d(o,q) + \max_{o_1,o_2\in G} d(o_1,o_2)\\
dist_{MinMax}(G,q) &= \min_{o\in G} d(o,q) + \max_{o_1,o_2\in G} d(o_1,o_2)\\
dist_{MinMax2}(G,q) &= \max\{\min_{o\in G} d(o,q), \max_{o_1,o_2\in G} d(o_1,o_2)\}
\end{aligned}$$

Note that the above distance function definitions are simplified, with some tunable parameters omitted due to page limit. We refer readers to the original papers [4] for the full list and complete definitions.

Different distance functions can be used to capture different needs. For example, $dist_{MaxSum}$ can fit with applications such as tourists planning visits to multiple POIs, and concern the farthest distance they have to travel from the hotel, as well as the distance between POIs. On the other hand, $dist_{MinMax}$ is useful when the user want to reach their first stop quickly, and explore other objects within a small region.

CSKQS also supports a variant of CoSKQ, called *Cost-Aware and Distance-Constrained Collective Spatial Keyword Query*, which is defined as follows.

*Problem 2 (CD-CoSKQ [3]):* 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. □

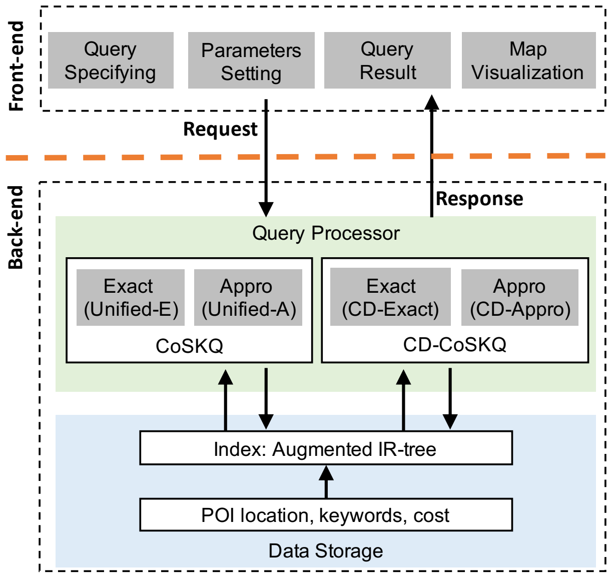


Fig. 1. System architecture of CSKQS

Compared to the conventional CoSKQ that find the feasible set with minimum $dist(G, q)$, the CD-CoSKQ problem is to find a feasible set $G$ which has its distance $dist(G, q)$ at most a threshold $B$ and its cost $cost(G)$ as small as possible.

**Cost Functions.** We consider two cost functions $cost(G)$.

$$\begin{aligned}
cost_{Max}(G) &= \max_{o\in G} o.w\\
cost_{Sum}(G) &= \sum_{o\in G} o.w
\end{aligned}$$

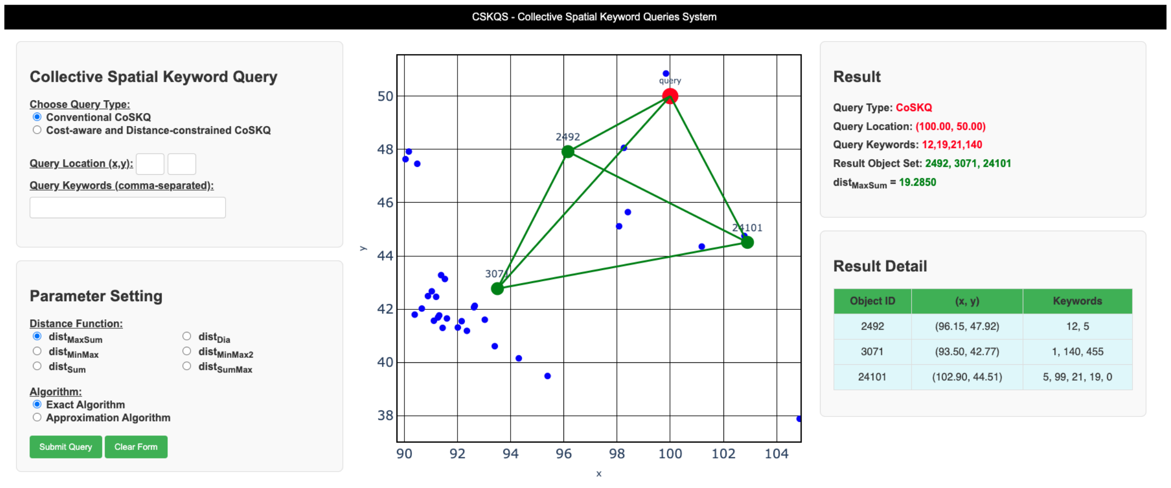
The cost function $cost_{Max}(G)$ is suitable for cases where costs represent levels of dissatisfaction, such as user ratings, with the highest cost reflect the maximum acceptable level of the dissatisfaction for the object set. In contrast, $cost_{Sum}(G)$ captures the total costs of the objects in $G$, which is suitable for cases when the costs represent some form of expense, such as time or money.

## III. CSKQS Design

In this section, we present the design of CSKQS. We first give an overview of the system architecture. Then, we introduce different components in the system.

### A. System Architecture

Figure 1 shows the system architecture of CSKQS. It adopts a front-end and back-end architecture. The front-end is a user interface to (1) allowing users to input queries and set parameters, then sending them to the back-end server, and (2) receiving the results from the server and presenting them to the users. The back-end consists of two main layers, namely the Query Processor and Data Storage. The Query Processor



CSKQS - Collective Spatial Keyword Queries System

Collective Spatial Keyword Query

Choose Query Type:
[✓ Conventional CoSKQ]
Cost-aware and Distance-constrained CoSKQ

Query Location (x,y):

Query Keywords (comma-separated):

Parameter Setting

Distance Function:
[✓ $dist_{MaxSum}$] $dist_{Dia}$
$dist_{MinMax}$ $dist_{MinMax2}$
$dist_{Sum}$ $dist_{SumMax}$

Algorithm:
[✓ Exact Algorithm]
Approximation Algorithm

Result

Query Type: CoSKQ
Query Location: (100.00, 50.00)
Query Keywords: 12,19,21,140
Result Object Set: 2492, 3071, 24101
$dist_{MaxSum}$ = 19.2850

Result Detail

| Object ID | (x, y) | Keywords |
|---|---|---|
| 2492 | (96.15, 47.92) | 12, 5 |
| 3071 | (93.50, 42.77) | 1, 140, 455 |
| 24101 | (102.90, 44.51) | 5, 99, 21, 19, 0 |

Fig. 2. User Interface of CSKQS

receives the input queries from the front-end, and retrieve the relevant data from the Data Storage.

## B. Query Processor

The query processor is designed to handle both the CoSKQ and CD-CoSKQ. The corresponding exact or approximation algorithm is invoked based on the user's input. The algorithm utilizes the index to retrieve the data, and returns the solution to the client-side. It is worth noting that the two problems requires different search procedures and algorithmic designs. Thus, the algorithms developed for answering CoSKQ cannot be applied to CD-CoSKQ, and vice versa.

- For CoSKQ, the Unified-E and Unified-A [4] are used as the exact and approximation algorithms, respectively.
- For CD-CoSKQ, the CD-Exact and CD-Appro [5] are used as the exact and approximation algorithms, respectively. CD-Appro is an $(\alpha, \beta)$-approximation algorithm, where the returned object set has it cost at most $\alpha$ times from the minimum cost of any feasible set satisfying the distance budget, and its distance within a factor of $\beta$ from the distance budget.

The approximation ratios of both Unified-A and CD-Appro depend on the specific distance function and cost function used. For example, when $dist_{MinMax}$ is used, Unified-A is a 2-approximation algorithm. When $cost_{Max}(\cdot)$ and $dist_{MaxSum}(\cdot)$ are used, CD-Appro is a $(1, 1.375)$-approximation algorithm.

## C. Indexing

To allow fast retrieval of spatial objects, we adopt the IR-tree [6] for keyword-based nearest neighbour queries and range queries, which are procedures invoked in the search algorithms. 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 better suit the algorithms, 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. Thus, the inverted lists not only store the nodes in the sub-tree that contain the corresponding keywords, but also an additional value, the minimum weight of the objects under each node.

## IV. Demonstration of CSKQS

We introduce the user interface of CSKQS and then detail the demonstration scenario in this section. In addition, an introduction video is available[1]. The user interface and the back-end are implemented in Python and C++, respectively.

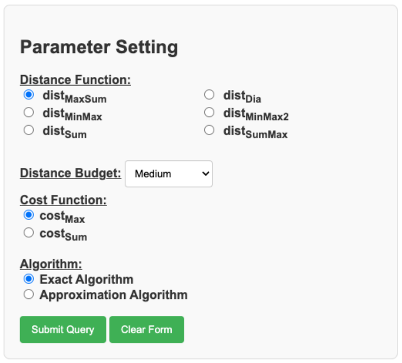
We extracted data from Yelp Open Dataset[2] for this demonstration. The dataset contains real-world POIs (i.e., objects), each has a spatial location, a rating on a 5-star scale with 0.5-star increments, and belongs to a set of business categories (e.g., Pubs, Burgers). For each object, the set of categories are used as its keywords, and the rating is converted to the range of [1,10], where lower is better. Note that we use the keyword ids, instead of textual keywords, for demonstration purpose.

## A. User Interface

Figure 2 shows the user interface of CSKQS, which contains five components as follows.

1) *Query Inputs (Top-left).* The users can input the query type, query location and query keywords. The users need to choose the query type to be either CoSKQ or CD-CoSKQ. For the query location, users can input $(x, y)$, or get the location from the map directly. For the query keywords, the users can input multiple keywords, separated by comma.
2) *Parameter Settings (Bottom-left).* The users can customize their query parameters by selecting the distance function

[1]https://youtu.be/QtAYUGd92u8
[2]https://business.yelp.com/data/resources/open-dataset



**Parameter Setting**

Distance Function:
[✓ $dist_{MaxSum}$] $dist_{Dia}$
$dist_{MinMax}$ $dist_{MinMax2}$
$dist_{Sum}$ $dist_{SumMax}$

Distance Budget: Medium

Cost Function:
[✓ $cost_{Max}$]
$cost_{Sum}$

Algorithm:
[✓ Exact Algorithm]
Approximation Algorithm

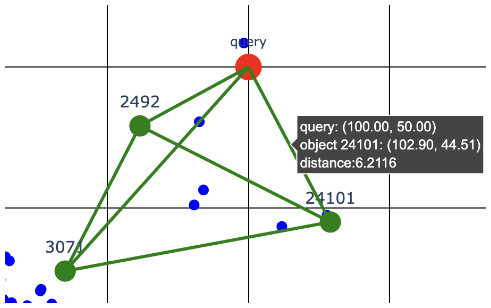
Fig. 3. Parameter Settings for CD-CoSKQ

and the algorithm to be used. In case of CD-CoSKQ, the users can also select the cost function and set the distance budget, as shown in Figure 3. Inspired by [8] that enforces budget constraints on route search, CSKQS provides five options {"Very Strict", "Strict", "Medium", "Loose", "Very Loose" } for users to choose their distance budget. It corresponds to $n = \{1.05, 1.1, 1.15, 1.2, 1.25\}$, respectively, where $n$ is the multiple that the user can tolerance compared to the result of the conventional CoSKQ. Users can also choose to input a real value to be the distance budget. These settings allow users to tailor queries to suit their personal needs and preferences.

3) *Interactive Map Visualization (Center).* This component enables users to interact with the map, performing operations like zooming in/out, panning to view different areas that they are interested in. Initially, the map mainly shows the objects' spatial distribution, where the objects are colored in blue. The users can look at the detail of each object, including its id, precise location, and associated keywords, by hovering the mouse cursor over the points. Once the user submits a query, the map highlights the query location in red color, and all the objects in the returned object set $G$ in green. It also shows the pairwise lines connecting the objects in $G$ and query location. By hover over these lines, the user can view the corresponding specific distance values, as shown in Figure 4.
4) *Query Results (Top-right).* This area displays the result of the query. It lists the object IDs in the returned object set $G$, along with the distance function value $dist(G)$, and in the case of CD-CoSKQ, the cost function value $cost(G)$ is also shown.
5) *Result Details (Bottom-right).* Finally, this area provides the returned object set information in detail, including each object's id, location, and the associated keywords listed in a table.

## B. Demonstration Scenario

We show the following three steps for a user to configure and submit a query, and view the results.



query: (100.00, 50.00)
object 24101: (102.90, 44.51)
distance:6.2116

Fig. 4. Hover text of the pairwise line

1) **Input Query.** First, the user chooses the query type from CoSKQ or CD-CoSKQ. The user then inputs the query location $(x, y)$ in the input box, and specifies the query keywords in the input box. Multiple keywords are separated by comma.
2) **Set Parameters.** The user selects the distance function to be $dist_{MaxSum}$, $dist_{Dia}$, $dist_{Sum}$, $dist_{SumMax}$, $dist_{MinMax}$, or $dist_{MinMax2}$. In case of CD-CoSKQ, the user also specifies the cost function to be $cost_{Max}$ or $cost_{Sum}$, and the distance budget. The user then selects to run either the exact or the approximation algorithm.
3) **View Results.** The user views the returned object set in the result box, along with its distance function value, and its cost function value in the case of CD-CoSKQ. The user can browse the object details information in the result detail table. The map visualizes the object set result in green, and helps the user understand the spatial relationship between the objects and query location. The user can then use the interactive map to zoom in to the region with object set, and move the mouse over the lines to see the spatial relationship and distance between the points. Users can easily see how the *distance contributor objects* [4] contributes to the distance function value.

## REFERENCES

[1] 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.
[2] X. Cao, G. Cong, C. S. Jensen, and B. C. Ooi. Collective spatial keyword querying. In *SIGMOD*, pages 373–384. ACM, 2011.
[3] H. K.-H. Chan, S. Liu, C. Long, and R. C.-W. Wong. Cost-aware and distance-constrained collective spatial keyword query. *TKDE*, 35(2):1324–1336, 2021.
[4] H. K.-H. Chan, C. Long, and R. C.-W. Wong. Inherent-cost aware collective spatial keyword queries. In *SSTD*, pages 357–375, 2017.
[5] H. K.-H. Chan, C. Long, and R. C.-W. Wong. On generalizing collective spatial keyword queries. *TKDE*, 30(9):1712–1726, 2018.
[6] 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.
[7] Y. Li, S. Yang, M. A. Cheema, Z. Shao, and X. Lin. Indoorviz: A demonstration system for indoor spatial data management. In *SIGMOD*, pages 2755–2759, 2021.
[8] T. Liu, Z. Feng, H. Li, H. Lu, L. Shou, and J. Xu. Ikaros: An indoor keyword-aware routing system. In *ICDE*, pages 3182–3185. IEEE, 2022.
[9] 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.
[10] C. Luo, Q. Liu, Y. Gao, L. Chen, Z. Wei, and C. Ge. Task: An efficient framework for instant error-tolerant spatial keyword queries on road networks. *PVLDB*, 16(10):2418–2430, 2023.
[11] J. X. Yew, N. Liao, D. Mo, and S. Luo. Example searcher: A spatial query system via example. In *ICDE*, pages 3635–3638. IEEE, 2023.