CSQ System: A System to Support Constrained Skyline Queries on Transportation Networks

Qixu Gong
Computer Science
New Mexico State University
Las Cruces, New Mexico
qixugong@nmsu.edu

Jiefei Liu
Computer Science
New Mexico State University
Las Cruces, New Mexico
jiefei@nmsu.edu

Huiping Cao
Computer Science
New Mexico State University
Las Cruces, New Mexico
hcao@nmsu.edu

***Abstract*—Skyline queries find the representative data points from a multi-dimensional dataset, which are better than other data points on at least one dimension. A multi-cost transportation network (MCTN) can be modeled as a multi-dimensional dataset. The MCTN-constrained skyline query (CSQ) is a type of skyline queries on MCTN where the query point and the skyline answer-objects are points of interest (POI) that are *off* the network, and the answer-points need to be reached from the query point by utilizing the MCTN. CSQ is useful in many applications such as trip planning and apartment selection. For example, when a person wants to find an apartment, he/she may consider not only the price and the number of rooms of the apartment but also the cost and travel time of using public transportation from his apartment to his/her office.**

In this paper, we present a system to answer MCTN-constrained CSQs, namely CSQ system. This system is implemented as a web application, which allows users to input a query point from a web interface, get the skyline result by using several algorithms, and display the result on the web interface. We load the POIs and public transportation networks of three cities (Los Angeles, San Francisco, and New York) to the system, and explain how users can interact with the CSQ system.

***Index Terms*—Skyline queries, Transportation networks, Multi-dimensional data**

I. Introduction

The skyline query, first introduced in [2], is an important and useful operator in finding representative objects from a multi-dimensional dataset. The answer to a skyline query consists of objects that are better than other objects on at least one dimension. The brute-force method to find skyline answer-objects from a given dataset is very expensive. Many studies on skyline queries focus on utilizing the characteristics of the data to speed up the query process. Other studies attempt to reduce the size of the skyline answer-set by customizing skyline queries.

Conducting skyline queries on multi-cost networks (MCNs) has been studied [4]–[7] in recent years. Two examples of MCNs are road networks and multi-cost transportation networks (MCTNs). In an MCN, the weights of the edges are multi-dimensional. The skyline queries on MCTNs are more complex than traditional skyline queries because both the non-spatial and the spatial attribute information of the data points, and the spatial relationship between data points are considered. They also provide a more general perspective than queries on graphs with single-dimensional weights. In real applications (e.g., trip planning), the decision is made on multiple factors (or dimensions) such as the price, travel time, and travel cost.

The MCTN-constrained skyline query, MCTN-CSQ or CSQ for short, is a new type of skyline queries on an MCTN [3]. Given an MCTN G, where the edge weights have d_G dimensions, a set of points of interest (POIs) D, where each POI has d_D non-spatial properties, and a query object q, CSQ returns the set of skyline answer-objects $\mathcal{R} \subset D$. Note that the object (or POI) $o \in D$ can be on or off the MCTN G. The cost of any skyline answer-object is less than other objects not in $\mathcal{R}$ on at least one dimension. The cost of a skyline answer-object $\mathcal{R}$ consists of the properties of its corresponding POI $o \in D$ and the cost of a graph path when traveling from q to o through the MCTN G.

For example, to find a good and cheap apartment that can be reached by taking a bus from a company, we need to consider not only the apartment's properties, but also the cost of the MCTN path. Fig. 1 shows that the properties of an apartment are its price and ranking, while the costs of the graph path from q to o are the walking distance, and the travel time and cost using the MCTN. The overall cost of an apartment o contains the union of the apartment property and the graph path cost.

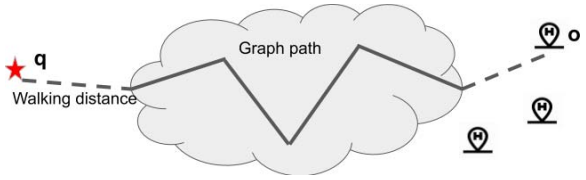


Fig. 1. Example of MCTN-CSQ: the cost of a skyline answer-object consists of the graph path cost (walking distance denoted using dash lines, travel fare, travel time) and the object's property (price, the environment)

MCTN-CSQ is very useful for applications such as trip planning and apartment selection.

No existing system can support CSQ on MCTN yet. We design a new system (CSQ system) to support CSQs. The CSQ system is implemented using an interactive map-based Web interface. It has the following functionality. (i) Answering CSQ: Our CSQ system provides a web interface to enable a user to input a customized CSQ. The system processes the CSQ and returns the skyline answer-objects for query q. (ii)

2375-026X/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDE48307.2020.00160

Understanding CSQ result: The CSQ result consists of the skyline POIs and the paths from the query point to these POIs. The system provides a friendly interface for users to examine those skyline answer-objects and the paths to reach those objects using MCTN. (iii) Comparing CSQ query processing algorithms: The system has implemented several algorithms to process the CSQ queries. Users can compare the effectiveness of the algorithms.

This CSQ system is built upon our work [3], which is the first work on solving the MCTN-CSQ problem. In [3], we propose an exact search algorithm (based on best-first search), two heuristic approximate approaches, and an index structure to process CSQs. The CSQ system has added **two new features** to make the system more friendly and to execute the algorithms more efficiently (Section II-C).

The paper is organized as follows. Section II presents the architecture of the CSQ system. Section III explains the usage of the system using a real-life example and compares the results returned by the exact search algorithm and the approximate methods. Section IV concludes this work.

II. Overview of the CSQ System

A. System Architecture

Fig. 2 shows the architecture of the CSQ system. The CSQ system consists of three major components. (i) The data storage and indexing component stores the POIs and the public transportation networks. (ii) The query processing component implements the exact and approximate search algorithms to answer CSQs. (iii) The web interface provides an interactive user interface (UI) to allow users to input queries and other related parameters, and also to visualize the CSQ answers.

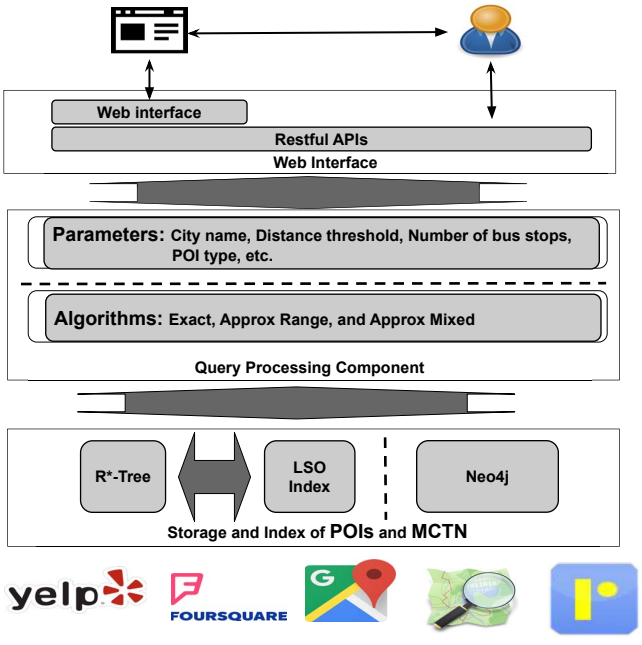


Fig. 2. Architecture of the CSQ system

B. Data Storage and Indexing Component

The CSQ system manages two types of data, the POIs D and MCTN G. Examples of POIs are restaurants and hotels that can be found from Yelp and Foursquare. Examples of MCTN are bus and metro lines downloaded from Google Maps, OpenStreetMap, and Rideschedules.

We utilize the R^* tree [1] and a Neo4j[1] graph database to store D and G respectively. R^* tree is a well-known spatial index which can be used to support different types of spatial queries. The set D contains all the POI objects. For each POI object, we keep its non-spatial attributes (e.g., price and review level of a hotel) and its location. The objects $o \in D$ with their non-spatial attribute values are indexed using an R^* tree. This is used to support comparisons over the non-spatial attributes, which are needed in finding skyline answer-objects. The spatial information of objects in D are not indexed.

Public transportation networks (e.g., bus line information) are stored using Neo4j. Neo4j is a widely used graph database management system. It can store graph nodes, edges, and properties efficiently. Neo4j provides APIs in different programming languages. These APIs enable users to implement many different customized graph algorithms. In the graph database, each vertex $v \in G$ represents a bus stop or a metro station in MCTN. Its spatial information (latitude and longitude) is stored as vertex properties in the graph database. An edge $e = (v_i, v_j)$ represents that there exists a bus line (or a metro line) segment that can go from a bus stop (or station) v_i to another bus stop (or station) v_j. Each edge has multi-cost weights (e.g., the fare and the travel time) which are stored as edge properties in the graph database.

The search algorithms in the query processing component (details see Section II-C) need to expand graph paths to get the final skyline answer-objects. The expansion of a graph path happens at the ending node of this path. The graph path expansion is prohibitively expensive. To accelerate this step, the search algorithm is designed to prune POIs that are not possible to be a skyline answer-object. To improve the pruning efficiency, the CSQ system builds an index, Local Skyline Objects (LSO) index, to organize the candidate skyline answer-objects for the ending nodes of the graph paths that need to be expanded. The detailed structure and design of the LSO index can be found from [3].

C. Query Processing Component

The query processing component implements the optimized exact search algorithm and two approximate algorithms proposed in [3].

The exact search algorithm can find all the skyline answers. This algorithm first identifies all the possible graph nodes that can be the starting node of a graph path. Then, it expands the graph nodes to graph paths (paths over MCTN) that are not dominated by any other paths. From the ending nodes of the graph paths, skyline objects can be reached. The number of the graph paths is exponential to the length of the paths. Thus,

[1]https://neo4j.com/

this approach is very inefficient and not practical to be utilized in real situation.

The two heuristic algorithms that return approximate solutions improve the performance of the exact search method. The first heuristic method is called **Range Approximate Method**. It finds approximate skyline answers by constraining the distance from the query point to the starting graph node and the distance from the ending graph node to the candidate skyline answer-object. The second heuristic method is called **Mixed Approximate Method**. It builds upon the ranged approximate method and further reduces the skyline answers by constraining the graph paths that need to be expanded. In particular, a graph path is expanded only when one of its weight values is minimal.

Besides implementing the presented algorithms in [3]. The CSQ system implements **two new features** to make it more friendly. First, the CSQ system allows the query point q to be any point that a user can specify on a map (e.g., by clicking a map location or by inputting an address). This generalizes the setting of [3], in which a query point needs to be an object in D. This generalization makes it more challenging to answer CSQs. When $q \in D$, we can use the properties of q to prune POIs that are dominated by q, which reduces the search space. However, when q is generalized, its properties are not in the database. Thus, it cannot be used to prune any POI. We address this issue by assuming that all POIs to be candidates for skyline answer-objects.

Second, the CSQ system adds one more constraint to the search algorithms by constraining the length of graph paths. This constraint is introduced to increase the system's usability in real life. In big cities, the above two heuristic methods still generate many (e.g., several hundreds) skyline answer-objects for a given query. The large answer set may confuse users. Also, displaying all these objects together messes up the interface. Introducing this constraint not only speeds up the query process, but also generates more decent skyline answer-set. This constraint can be incorporated with the exact search and two approximate search methods.

D. Web Interface

The user interface accepts queries with parameters and visualizes the skyline answers. A query point q and the query processing methods are two mandatory parameters. The query point can be entered in four different ways. (i) q can be selected from the list of D. (ii) q can be the user's current location (when it is available) represented with coordinates. (iii) q can be chosen by directly clicking the map (the coordinates of q are captured). (iv) q can be entered as an address, which is converted to the corresponding coordinates by the Google maps geocoding APIs. When the coordinates of q are provided and the distance between q and an object $o \in D$ is less than 30 meters, the CSQ system assumes that the query is equivalent to o. The query method is chosen from *the Mixed Approximate*, *the Range Approximate*, and *the Exact improved Index*. These methods all utilize the LSO index.

TABLE I
ACCEPTED PARAMETERS

Parameter	Description
City name	The city where the query happens
POI Type	The type of the desired skyline answer-objects
Distance threshold	The maximum distance that a user is willing to walk from the query point to the starting bus stop/metro station and from the ending bus stop/metro station to a target object
Number of stops	The maximum length of a graph path

Besides the two mandatory parameters, a user can input four optional parameters, which are listed in Table I. The parameter *City name* is used to narrow the search space in a reasonable manner. The parameter *POI type* captures the type of the user's target object. The parameters, *distance threshold* and *number of stops*, are used in the two approximate search methods. When the number of stops is not provided, the new constraint of limiting the length of graph paths is not applied.

The skyline answer is a set of objects of the given POI type. For each skyline answer-object, there may be multiple paths to reach it. To display these two parts of information, the CSQ system adds a result review panel to the left of a Google Map interface. When the query processing component returns an answer set, The skyline answer-objects are listed at the result view panel and are marked on the map. When a user clicks one skyline answer-object, the paths following which a user can reach this object from the query point are highlighted. The paths comprise three parts: (i) the walking path from the query point to the starting node v_s of the graph path, (ii) the graph path from v_s to its ending node v_t, (iii) and the walking path from v_t to the skyline answer-object.

When a skyline answer-object is clicked, a pop-up window shows its information, which includes its non-spatial properties, the total walking distance, the estimated travel fare, the total travel distance and travel time.

Besides the web interface, restful web service APIs are also provided. A user can directly use the restful URL to get the results in JavaScript Object Notation (JSON) format.

III. DEMONSTRATION

This section demonstrates the usage of the CSQ system by utilizing a real-life scenario, and compares the results generated by two different methods. For this demo, we preload the POIs and the public transportation networks in three cities, New York (NY), Los Angeles (LA), and San Francisco (SF), to the system. The dataset D contains 25,854 POIs (14,155 for LA, 2,110 for SF, and 9,589 for NY respectively). For each POI, we extract its location information and three non-spatial attributes (rating, price, and interestingness). For the transportation networks, we extract 5,127 nodes and 11,152 edges for NY, 9,041 nodes and 13,615 edges for SF, and 12,433 nodes and 22,752 edges for LA, from the https://rideschedules.com site.

A. Real-life Scenario

Let us consider a real-life application scenario. Alice attends a conference in San Francisco and wants to find a hotel with

a reasonable price and good service. The conference venue is treated as a query point, and the hotel that she wants to find is one skyline answer-object. Alive has more constraints about the desired hotel: it can be reached by taking public transportation, and the transportation fare and the travel time should be reasonable. Also, Alice does not want to take a bus for too long (e.g., at most ten stops), and she does not want to walk too much from the conference venue to the starting bus stop and from the ending bus stop to the desired hotel.

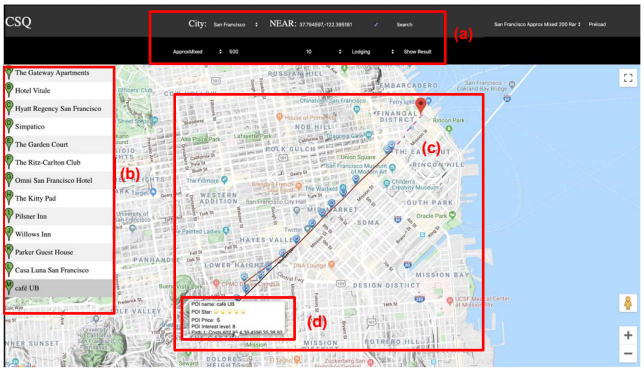
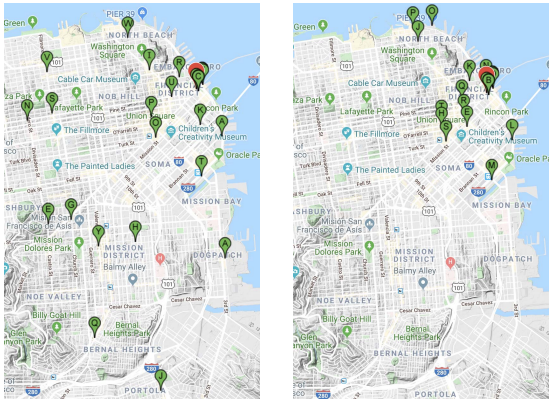


Fig. 3. The Query Process

Fig. 3 shows how Alice utilizes the CSQ system to complete the query and find her desired hotel. First, Alice identifies the query point as the conference venue by clicking the location of the conference venue or finding the conference venue from the list of D, chooses the search method, inputs the other parameters (e.g., city name is San Francisco, POI type is hotel, number of bus stops is 10), and click *show results* (Fig. 3(a)). Then, the skyline answer-objects (the possible hotels) are returned, listed at the left panel (Fig. 3(b)), and pined as green markers on the map. Next, Alice can click one hotel from the left panel, the map interface shows the public transportation routes to reach this hotel from the conference venue 3(c). The dashed lines represent the walking route from the conference venue to the starting bus stop and from the ending bus stop to the target hotel. The solid lines display the bus/metro route. The blue markers connected by the solid lines are the bus stops or metro stations. The different paths from the query point to the same object are shown in different colors. The information of a hotel is shown in a pop-up window (Fig. 3(d)). It tells Alice the properties of the hotel and the cost (including travel time and expense) of each path.

B. Comparison of Exact and Approximate Search Algorithms

The running time of the different algorithms is compared and reported in [3]. This demo focuses on examining the meaningfulness of the skyline results. In particular, we show how scattered the skyline answer-points are. Fig. 4(a) shows the results from the exact search algorithm. The skyline answer-points are scattered in almost the whole map of SF. We notice that some results are not reasonable. For example, the skyline answer-objects Q and J are too far away from the query point, which can be seen from the figure. Furthermore,



(a) Exact Method on SF (b) Approx Range Method on SF

Fig. 4. The comparison of the exact method and heuristic approaches

the user needs to walk a very long distance to Q and J (6380 meters and 7203 meters respectively), which is not shown in the figure due to space limitation.

Fig. 4(b) shows the results returned by the *Range approximate* method with same query setting. The results are more meaningful: the skyline objects are scattered in an area closer to the query point. The properties of each skyline answer-object are reasonable. When the maximum number of bus stops/metro stations is given, the system responds much faster and the skyline answer-set is more concise.

IV. Conclusions & Proposal

We implement the CSQ system to answer the MCTN-constrained CSQs, which was first proposed by us in [3]. This work demonstrates the CSQ system. The CSQ system preloads data (POIs and public transportation networks) of three cities (LA, NY, and SF) and can support query processing using one exact and two approximate search algorithms. Using this CSQ system, attendees can test answering CSQs in the above three cities. The demonstration can help participants understand the skyline answers shown on the list panel (skyline answer-objects) and the map panel (graph paths). Based on the understanding of the results, users can choose the desired POI.

References

[1] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The R*-tree: an efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, volume 19, pages 322–331, 1990.

[2] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430. IEEE, 2001.

[3] Qixu Gong, Huiping Cao, and Parth Nagarkar. Skyline queries constrained by multi-cost transportation networks. In *ICDE*, pages 926–937. IEEE, 2019.

[4] Hans-Peter Kriegel, Matthias Renz, and Matthias Schubert. Route skyline queries: A multi-preference path planning approach. In *ICDE*, pages 261–272. IEEE, 2010.

[5] Kyriakos Mouratidis, Yimin Lin, and Man Lung Yiu. Preference queries in large multi-cost transportation networks. In *ICDE*, pages 533–544. IEEE, 2010.

[6] Michael Shekelyan, Gregor Jossé, and Matthias Schubert. Linear path skylines in multicriteria networks. In *ICDE*, pages 459–470. IEEE, 2015.

[7] Bin Yang, Chenjuan Guo, Christian S Jensen, Manohar Kaul, and Shuo Shang. Multi-cost optimal route planning under time-varying uncertainty. In *ICDE*, 2014.