SkyEngine: Efficient Skyline Search Engine for Continuous Skyline Computations

Yu-Ling Hsueh [†1], Roger Zimmermann [*2], Wei-Shinn Ku [#3], Yifan Jin [§4]

[†]*Teradata, San Diego, CA, USA*
[1]`yuling.hsueh@teradata.com`

[*]*Computer Science Department, National University of Singapore, Singapore*
[2]`rogerz@comp.nus.edu.sg`

[#]*Dept. of Computer Science and Software Engineering, Auburn University, USA*
[3]`weishinn@auburn.edu`

[§]*Dept. of Computer Science, University of Hong Kong, China*
[4]`yfjin@cs.hku.hk`

Abstract— **Skyline query processing has become an important feature in multi-dimensional, data-intensive applications. Such computations are especially challenging under dynamic conditions, when either snapshot queries need to be answered with short user response times or when continuous skyline queries need to be maintained efficiently over a set of objects that are frequently updated. To achieve high performance, we have recently designed the *ESC* algorithm, an Efficient update approach for Skyline Computations. *ESC* creates a pre-computed *candidate skyline* set behind the first skyline (a "second line of defense," so to speak) that facilitates an incremental, two-stage skyline update strategy which results in a quicker query response time for the user. Our demonstration presents the two-threaded *SkyEngine* system that builds upon and extends the base-features of the *ESC* algorithm with innovative, user-oriented functionalities that are termed *SkyAlert* and *AutoAdjust*. These functions enable a data or service provider to be informed about and gain the opportunity of automatically promoting its data records to remain part of the skyline, if so desired. The *SkyEngine* demonstration includes both a server and a web browser based client. Finally, the *SkyEngine* system also provides visualizations that reveal its internal performance statistics.**

I. Introduction

Skyline queries have a rich history in the context of spatio-temporal databases and they are an important component of multi-criteria decision making applications. Skyline points may be defined as the set of objects which are not *dominated* by any other points. An object p dominates q, if p has equal or better attribute values than q in all dimensions and a better value than q in at least one. Most of the prior work on skyline query processing has assumed that data objects are static [1]. Other approaches have assumed that the skyline computations involve only a subset of the dynamic dimensions [2]. In contrast, our approach is designed to support Efficient updates for continuous Skyline Computations over dynamic objects (*ESC* for short), where objects with d dynamic dimensions move in an unrestricted manner. Each dimension may represent spatial or non-spatial values. To achieve efficient continuous skyline computations the following challenges must be addressed: (a) an effective incremental result update mechanism is needed to provide a fast response time when reporting the current query results, and (b) an efficient strategy is required to reduce the search space dimensionality.

Existing methods [3], [4] generally compute a number of data point subsets, each of which is exclusively dominated by one skyline point. Therefore, when a skyline point moves or is deleted, only its exclusively dominated subset must be scanned, hence reducing the workload. However, the determination of such an exclusive data set is very computationally complex in higher dimensions and incurs a serious burden on the system in very dynamic environments. As a consequence, these systems are often unable to provide up-to-date query results with a quick response time. We have designed the *ESC* algorithm to address this issue by delegating the time-consuming skyline update computations to another, independent procedure, which is executed after the query processor returns the latest skyline query results. The key idea is to maintain a *candidate skyline* set 'behind' the main skyline as a collection of points that are pre-computed whenever a skyline point submits an update. With the knowledge of the candidate skyline set, the query result can be updated from within a limited search space and the expensive computations (e.g., searching for new candidates to substitute for a promoted candidate skyline point) can be decoupled from the update process.

Building upon the core *ESC* functionality, our demonstration system provides two innovative new features called *SkyAlert* and *AutoAdjust* – both of which are designed to benefit a service provider (e.g., a hotel business). A *SkyAlert* informs a provider when she is about to gain or lose an advantage over her competition. Furthermore, if a hotel administrator enables the *AutoAdjust* feature, the system automatically alters one or more attribute values (within an acceptable range pre-set by the provider) to promote the data point to become a skyline point, which likely increases its appeal to customers. On the other hand, when a skyline point degrades to a candidate skyline or regular data point, the system alerts the hotel administrator to monitor the market and possibly take

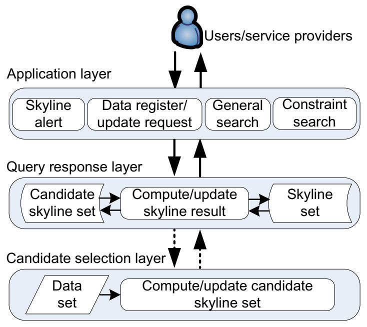


Fig. 1. System architecture overview.

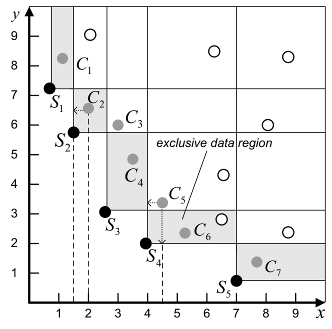


Fig. 2. Skyline points (black), candidate skyline points (grey), and regular data points (white).

action to attract more customers.

Figure 1 illustrates the block diagram of the *SkyEngine* system. A user query is accepted at the *application layer* (either a general or a constraint skyline search request). The *SkyAlert* module handles dimensional data adjustments and messaging between *SkyEngine* and the affected service providers. Each data update request from a service provider is also submitted to *SkyEngine* via the application layer. The request is then passed to the *query response layer*, where *SkyEngine* examines whether the update request (e.g., the insertion or deletion of a data point) affects the skyline set. If so, it performs the necessary result updates. In the case where an update request stems from a removed or moving skyline point, some exclusive points may be left un-dominated. Hence *SkyEngine* searches for new substitute skyline points only from within the candidate skyline set. The query results are immediately returned at the end of this task, which is performed in the query response layer. The processing time of the sequence of tasks in the application and the query response layer represent the system response time to a skyline query update. The *candidate selection layer* is responsible for determining and maintaining the candidate skyline set when any candidate skyline points are inserted or removed. To enhance this task, which involves the expensive computation of determining exclusive data points when new or substitute candidate skyline points are found, we employ an *approximate exclusive data region* (*AEDR*) computation with lower amortized cost than existing techniques [3], [4].

II. SkyEngine System Design

SkyEngine is implemented with a client-server architecture. The application layer manages the communication between the system and its clients. The query response layer and candidate selection layer form the core of *SkyEngine* to handle the back-end processes. The details of each layer are described in the following sections.

A. Application Layer

Queries from users and data point registrations/updates from service providers are handled by this layer. Four different types of queries are supported: a user can launch either a snapshot or continuous skyline search and both types can further be evaluated as either a constraint or a general search. A snapshot query request returns a set of static result points which are retrieved from the data set with the most recently updated values before the request was made. Conversely, with a continuous skyline query a set of dynamic skyline results are continuously displayed. Both snapshot and continuous queries can either consider the entire data set (general search) or only data tuples that satisfy certain criterias (constraint search). When a data register or update request is received in the application layer, procedures in the query response layer are triggered to update the skyline results based on the changes. To utilize the *AutoAdjust* and *SkyAlert* features, a service provider must specify a pair of values (*i.e.*, a default and a threshold – minimum or maximum – value) for each dimension of a data point. Initially, for each point, the system uses the default values for all dimensions when starting the skyline computations and automatically adjusts the attribute values up or down towards the threshold values when promoting a non-skyline point later. A default value would be more profitable for a service provider, while a threshold value (the least acceptable value for a service provider) is more attractive to a service customer. When *SkyEngine* performs the *AutoAdjust* procedure for a data point p, it locates the associated skyline point (the closest skyline point) of p and performs the minimal adjustments on the dimensions in user-defined order. For efficiency reasons the *SkyAlert* and *AutoAdjust* features are enabled only for candidate skyline points, because the adjustment cost of promoting a candidate skyline is cheaper than for a regular point.

Consider the example in Figure 2. To promote point C_2 (a candidate skyline point), the system reduces the value of the x dimension (the first priority dimension) by the minimal difference between $C_2.x$ and $S_2.x$, where S_2 is the associated skyline point of C_2. In this example, it is a one-step adjustment (a reduction of only $C_2.x$'s value by 0.5). The special case of multi-step adjustments for a non-exclusive candidate skyline point is also handled by the system. For example, the system first adjusts $C_5.x$ (associated with S_4) and then requires a second update for $C_5.y$ to promote C_5. Multi-step adjustments occur only when a candidate skyline point is not an exclusive data point in an *EDR*. In an extreme case a skyline point and a candidate skyline point may compete with each other for several rounds, but active adjustments are made only until one dimension of either of the two points reaches the pre-set threshold value. Finally, *SkyAlert* completes a procedure to trigger a message to inform the owner of either a skyline or a candidate skyline point of its current status.

B. *Query Response Layer*

The skyline result set is evaluated and progressively updated through the operations in the query response layer. The major challenge with continuous query processing is to avoid unnecessary dominance checks on irrelevant data points for skyline result updates. *SkyEngine* utilizes a candidate skyline set to achieve a much shorter response time for skyline updates compared with existing solutions [2], [3]. A point is a candidate skyline point if and only if it is not dominated by any other candidate skyline point, except for a skyline point. Figure 2 shows the skyline and candidate skyline points in a two-dimensional data set. The initial candidate skyline set can always be obtained with little extra work while computing the skyline points. When a skyline point s_i is removed or at least one value of its dimensions changes, the candidate skyline points are considered to determine whether they can "substitute" for s_i. The features of the candidate skyline set are as follows: (1) it is a pre-computed set that covers *all* the potentially new skyline points, and (2) it is a relatively small data set. Therefore, with the knowledge of the candidate skyline set, the query processor can efficiently update the query results and provide a quick response to the query requester. The formal definition and proof of the correctness of a candidate skyline set can be found in our prior work [5]. *SkyEngine* is implemented with an event-driven architecture to handle query updates. When the system receives a data update request (for an existing skyline point, a candidate skyline point, or a dominated data point), it first engages the query response layer to examine whether the request affects the skyline result set and it then outputs the updated skyline points if the result set has been modified. Next, the remaining non-skyline-related computations (*i.e.*, the evaluation and update of the candidate skyline set) are processed in the candidate selection layer.

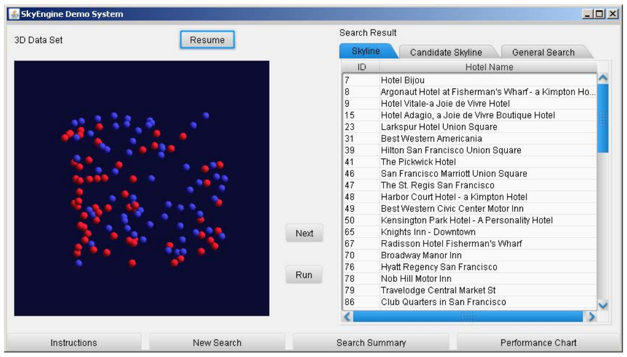
C. *Candidate Selection Layer*

SkyEngine delegates the most time-consuming computations to another, independent procedure performed in the candidate selection layer as a result of utilizing a candidate skyline set. Nevertheless, the candidate selection layer needs to ensure a timely and up-to-date candidate skyline set, which is critical for the correctness of query results. In the candidate selection layer, the candidate skyline updates are handled efficiently by adopting an *approximate exclusive data region* (*AEDR*) technique, which outperforms existing approaches [4], [3]. A new candidate skyline point is selected only from a corresponding *AEDR*. Prior techniques either compute traditional exclusive data regions (*EDRs*) or perform intersection checks without explicitly calculating the representation of an *EDR*. The general purpose of an *EDR* is to reduce the search space by establishing a number of data point subsets, each of which is exclusively dominated by one skyline point. Therefore, when a skyline point moves or is deleted, only its exclusively dominated subset must be scanned. In the existing work, the *EDR* computation is part of the skyline re-evaluation processes. Because of the computational complexity in higher dimensions (*i.e.*, an *EDR* becomes a complex irregular shape), users often encounter significant latency while waiting for the query results. Unlike the traditional *EDR* which is a minimal region that encloses the entire corresponding exclusive data points, an *AEDR* might be larger and thus include more data points in the region. However, it is always a regularly-shaped region and hence it achieves a lower amortized cost.

III. Demonstration

The *SkyEngine* server supports Web services which publish the major *SkyEngine* functions to the world. Users can access these functions through web-based applications that engage the *SkyEngine* functions. A result set is returned to the client application, which can then transform the transmitted results into a desirable format for presentation. *SkyEngine* runs on any PCs with Internet connectivity and its main data store is implemented with a *MySQL* database which contains two sample data sets. We have collected multi-dimensional data from a major on-line travel agency, Travelocity, for the hotel domain and retrieved stock market information from Google Finance, which provides a Market Data API, respectively. Figure 3(a) illustrates such a web-based user interface of *SkyEngine*. For example, a user may search for hotels within a 15-mile radius in the Los Angeles area. The left window frame visualizes all the data points (up to three dimensions) with the skyline objects denoted in red, and candidate skyline objects in blue (regular data points are omitted). When a data set contains more than three dimensions, we only project the first three. However, the skyline computation can handle high-dimensional data sets (more than $3D$). The coordinate system may be rotated by the user to generate a different viewing angle. If a user requests a continuous skyline query, all the data points are shown in motion to reflect their current data values in the computational space. In this demo system, we provide two execution modes: (1) a step-by-step mode that shows the query results once at a time by clicking the *Next* button and (2) a continuous execution mode that continuously shows the query result after clicking the *Run* button. In the right window frame of Figure 3(a), the skyline result points (e.g., hotels) are listed. To view the entire "unfiltered" result set returned by a non-skyline (traditional) query processor (e.g., all the available hotels within a 15-mile radius in the Los Angeles area), the user can click on the *General Search* tab in the right frame.

By clicking on the *Search Summary* button in the tool bar shown in Figure 3(a), a window frame as illustrated in Figure 3(b) is shown. It lists a summary of the current skyline points and users can also view the detailed information of each point with a mouse click. The summary search window presents the skyline results grouped by the number of superior attributes. By definition, a skyline point has at least one attribute whose value is not worse than for all the other skyline points. Therefore, *SkyEngine* presents up to $d - 1$ superior attributes in the summary window. A user may prefer a skyline point with multiple, rather than one, superior attributes.

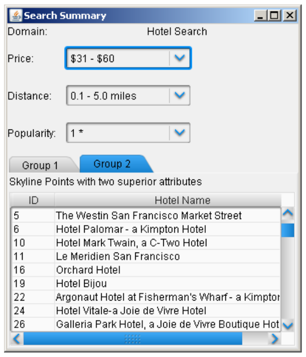


Search Result

Skyline | Candidate Skyline | General Search

ID	Hotel Name
7	Hotel Bijou
8	Argonaut Hotel at Fisherman's Wharf - a Kimpton Ho...
9	Hotel Vitale-a Joie de Vivre Hotel
15	Hotel Adagio, a Joie de Vivre Boutique Hotel
23	Larkspur Hotel Union Square
31	Best Western Americania
39	Hilton San Francisco Union Square
41	The Pickwick Hotel
46	San Francisco Marriott Union Square
47	The St. Regis San Francisco
48	Harbor Court Hotel - a Kimpton Hotel
49	Best Western Civic Center Motor Inn
50	Kensington Park Hotel - A Personality Hotel
65	Knights Inn - Downtown
67	Radisson Hotel Fisherman's Wharf
70	Broadway Manor Inn
76	Hyatt Regency San Francisco
78	Nob Hill Motor Inn
79	Travelodge Central Market St
86	Club Quarters in San Francisco

(a) *SkyEngine* user interface.



Domain: Hotel Search

Price: $31 - $60

Distance: 0.1 - 5.0 miles

Popularity: 1 *

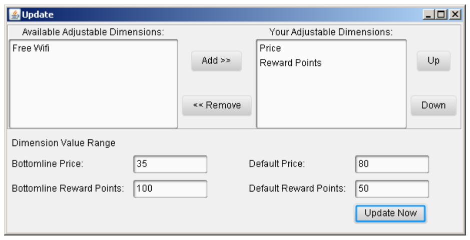
Skyline Points with two superior attributes

ID	Hotel Name
5	The Westin San Francisco Market Street
6	Hotel Palomar - a Kimpton Hotel
10	Hotel Mark Twain, a C-Two Hotel
11	Le Meridien San Francisco
16	Orchard Hotel
19	Hotel Bijou
22	Argonaut Hotel at Fisherman's Wharf - a Kimptor
24	Hotel Vitale-a Joie de Vivre Hotel
26	Galleria Park Hotel, a Joie de Vivre Boutique Hot

(b) Search summary panel.

Fig. 3. Information presented in the *SkyEngine* web browser client.

Furthermore, to help users narrow their choices among skyline result points, the interface provides additional advanced search functions for each attribute (e.g., to specify a price or distance range). In the *New Search* window, the users may specify a desirable search type as a combination of the following categories: (1) a continuous or (2) a snapshot skyline query, and (a) a general or (b) a constraint skyline query. A user may request a continuous, constraint skyline query, where the values of certain attributes are considered within a user specified range (e.g., price $<$ \$100) and the query results are continually posted to the user interface.



Available Adjustable Dimensions: Free Wifi

Your Adjustable Dimensions: Price, Reward Points

Dimension Value Range

Bottomline Price: 35 | Default Price: 80

Bottomline Reward Points: 100 | Default Reward Points: 50

Fig. 4. *SkyAlert* user interface.

A client interface in Figure 4 is designed for a service provider to update their customized data settings when it enables the *AutoAdjust* feature. For example, a hotel administrator can select one or more dimensions (e.g., price or reward points) that are feasible to adjust for their business from the left list panel. The adjustment sequence for each selected dimension in the right list panel can be set by clicking the up and down arrow to change the order. The input fields for each selected dimensions are provided at the bottom of the window. For example, the administrator may input a bottomline value for the price dimension and one for the reward points dimension, allowing the *AutoAdjust* feature to adjust the price up or down to this value when promoting the point. The default value is the initial value that a service provider would like to start with having the goal of gaining a better profit. Finally, in the *SkyEngine* system, we compare the response times for a new skyline query evaluation and for an existing skyline query update by visualizing a dynamic line chart in Figure 5 for our *ESC* algorithm, the well-known *BBS* skyline query approach [4] and the *DeltaSky* approach [3], which is a recent technique that handles continuous skyline queries by determining the *EDR* data sets without explicitly calculating the *EDR* itself. To achieve a fair comparison, we also enhance the *BBS* algorithm by adopting the *ABBS* (Adaptive Branch-and-Bound) to avoid complex irregular-shaped *EDR* computations. Figure 5 shows that *ESC* outperforms both methods in terms of the response time and overall CPU time.

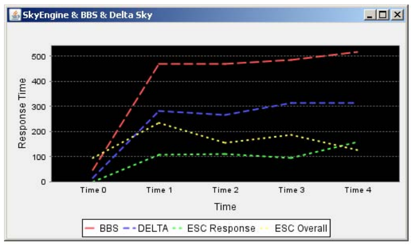


Fig. 5. Performance comparison chart.

IV. Conclusions

We have presented *SkyEngine*, a skyline query processing engine that is specifically designed to provide fast interactive response times to users. Its core *ESC* algorithm is based on an incremental skyline update mechanism. With the adoption of pre-computed candidate skyline sets, *ESC* efficiently updates the skyline query results and delegates the most complex computations to a separate procedure that executes after the updates of the query results are completed. The *AutoAdjust* and *SkyAlert* features provide a better solution for a service provider to promote its service points automatically and effectively. *SkyEngine* demonstrates the core *ESC* features and the auto-update/alert functions in a client-server system.

References

[1] J. Pei, W. Jin, M. Ester, and Y. Tao, "Catching the Best Views of Skyline: A Semantic Approach Based on Decisive Subspaces," in *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB), Trondheim, Norway*, 2005, pp. 253–264.

[2] Z. Huang, H. Lu, B. C. Ooi, and A. K. H. Tung, "Continuous Skyline Queries for Moving Objects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 12, pp. 1645–1658, 2006.

[3] P. Wu, D. Agrawal, Ö. Egecioglu, and A. E. Abbadi, "Deltasky: Optimal Maintenance of Skyline Deletions without Exclusive Dominance Region Generation," in *Proceedings of the 23rd International Conference on Data Engineering (ICDE),Turkey*, 2007, pp. 486–495.

[4] D. Papadias, Y. Tao, G. Fu, and B. Seeger, "Progressive Skyline Computation in Database Systems," *ACM Transactions on Database Systems*, vol. 30, no. 1, pp. 41–82, 2005.

[5] Y.-L. Hsueh, R. Zimmermann, and W.-S. Ku, "Efficient Updates for Continuous Skyline Computations," in *Proceedings of the 19th International Conference on Database and Expert Systems Applications (DEXA), Turin, Italy*, 2008, pp. 419–433.