Datenbank Spektrum (2016) 16:247–251
DOI 10.1007/s13222-016-0229-2

KURZ ERKLÄRT

Skyline Queries

Katja Hose[1]

Received: 16 May 2016 / Accepted: 22 June 2016 / Published online: 6 July 2016

Abstract Many applications face the problem that users are overwhelmed by the large amount of available data. In some cases an objective ranking function can be used to order data items by their relevance – similar to the top 10 results displayed by a Web search engine. Other applications, however, aim at considering more diverse preferences and multiple criteria to help users find good results. Such applications can benefit from skyline queries.

The best known example use case for a skyline query is a hotel booking scenario where users are looking for hotels. Assume many hotels are available and the user wants to find one based on two criteria: distance to the beach and price per night. Further assume that the user is unable to say which of these criteria is more important. So, we need to look for hotels representing a good combination of both criteria. The skyline consists of all hotels that represent a "good" combinations of both criteria. For each of the other hotels, there is always at least one hotel in the skyline that is better with respect to the two criteria. So, being presented the skyline, the user gets an overview of the available hotels and can make the final decision with respect to her personal preferences for the two criteria. No matter how the user will eventually weigh her personal preferences, she will find her favorite hotel in the skyline.

This article gives a short introduction to skyline queries, their main characteristics, and basic ways of processing them.

✉ Katja Hose
khose@cs.aau.dk

[1] Aalborg University, Aalborg, Denmark

1 Introduction

Many applications face the problem that users are overwhelmed by the large amount of available data. In some cases an objective ranking function can be used to order data items by their relevance – similar to the top 10 results displayed by a Web search engine. Other applications, however, aim at considering more diverse preferences and multiple criteria to help users find good results. Such applications can benefit from skyline queries.

The best known example use case for a skyline query is a hotel booking scenario where users are looking for hotels. Assume many hotels are available and the user wants to find one based on two criteria: distance to the beach and price per night. Further assume that the user is unable to say which of these criteria is more important. So, we need to look for hotels representing a good combination of both criteria – if the user was able to provide weights for both criteria, it might be possible to define a single ranking function and use a top-k query instead. Figure 1 shows the available hotels; each hotel is represented as a circle, the x-axis corresponds to the distance to the beach and the y-axis to the price. The hotels highlighted in red are part of the skyline and represent the complete set of "good" combinations of both criteria. For each of the other hotels, there is always at least one hotel in the skyline that is better with respect to the two criteria. Being presented the skyline, the user gets an overview of the available hotels and can make the final decision with respect to her personal preferences for the two criteria. No matter how the user will eventually weigh her personal preferences, she will find her favorite hotel in the skyline.

The problem of computing skyline queries is also known as the Pareto optimum or the maximum vector problem [5]. The term skyline query, however, was introduced

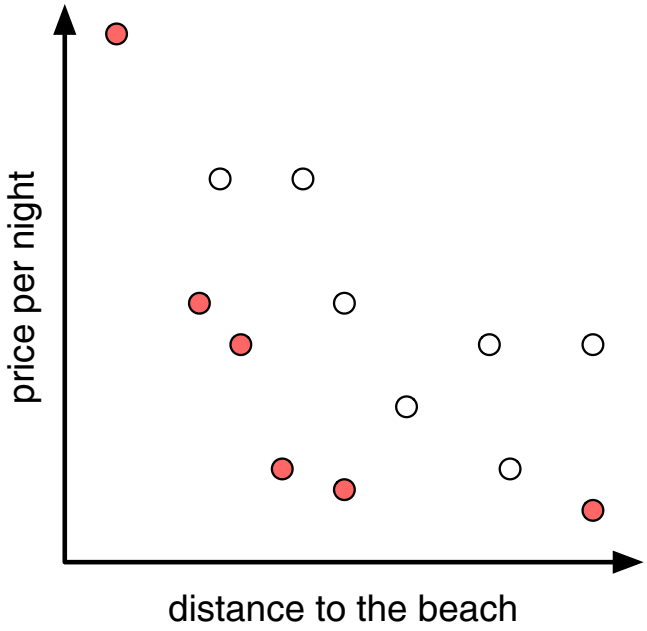

Fig. 1 Example Skyline

Fig. 2 Skyline of Singapore

by S. Börzsönyi, D. Kossmann, and K. Stocker in 2001 [1]. Skyline queries were proposed as a tool for multi-criteria decision making by using multiple ranking functions to find relevant data items for a user. Figure 2 shows the skyline of Singapore to illustrate why the term skyline query was chosen. In such a skyline, we can only see buildings that are very high or close, more precisely: we can only see buildings representing good combinations of the two criteria distance and height – roughly corresponding to distance and price in the hotel example. The term skyline was chosen because of its graphical representation.

In the following, we first present the main characteristics of a skyline and then discuss several types of skyline queries that have been proposed in the past 15 years. Afterwards, we will briefly sketch the basics of skyline query computation and recent trends.

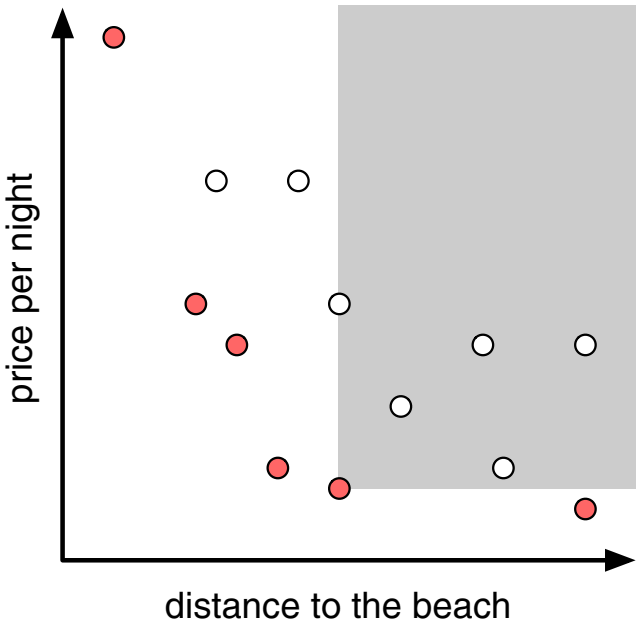

Fig. 3 Dominance

2 Characteristics

Formally, a skyline query is defined on dimensions representing the criteria that the user is interested in. The hotel example, for instance, has two dimensions: price and distance to the beach. A skyline can of course be defined on more dimensions. However, skyline queries are defined on at least two dimensions as having only one dimension would, for instance, correspond to simply finding the cheapest hotel.

For each dimension of a skyline, the user needs to specify whether small values (MIN) or big values (MAX) are desired. The original skyline paper [1] also proposed DIFF; indicating that it is sufficient to have a different value rather than a "better" value (smaller for MIN, greater for MAX). In the hotel example, MIN is used for both dimensions: the distance to the beach and the price. Combinations of MIN and MAX for different dimensions are of course possible.

One of the main principles that a skyline is defined on is the *dominance* relation: all data items in the skyline are not dominated by any other data item in the dataset. A data item dominates another one if it is as good as or better (with respect to MIN/MAX) in all dimensions and better in at least one dimension. Or in other words, a data item dominates another one if it is better in at least one dimension and not worse in all the other dimensions.

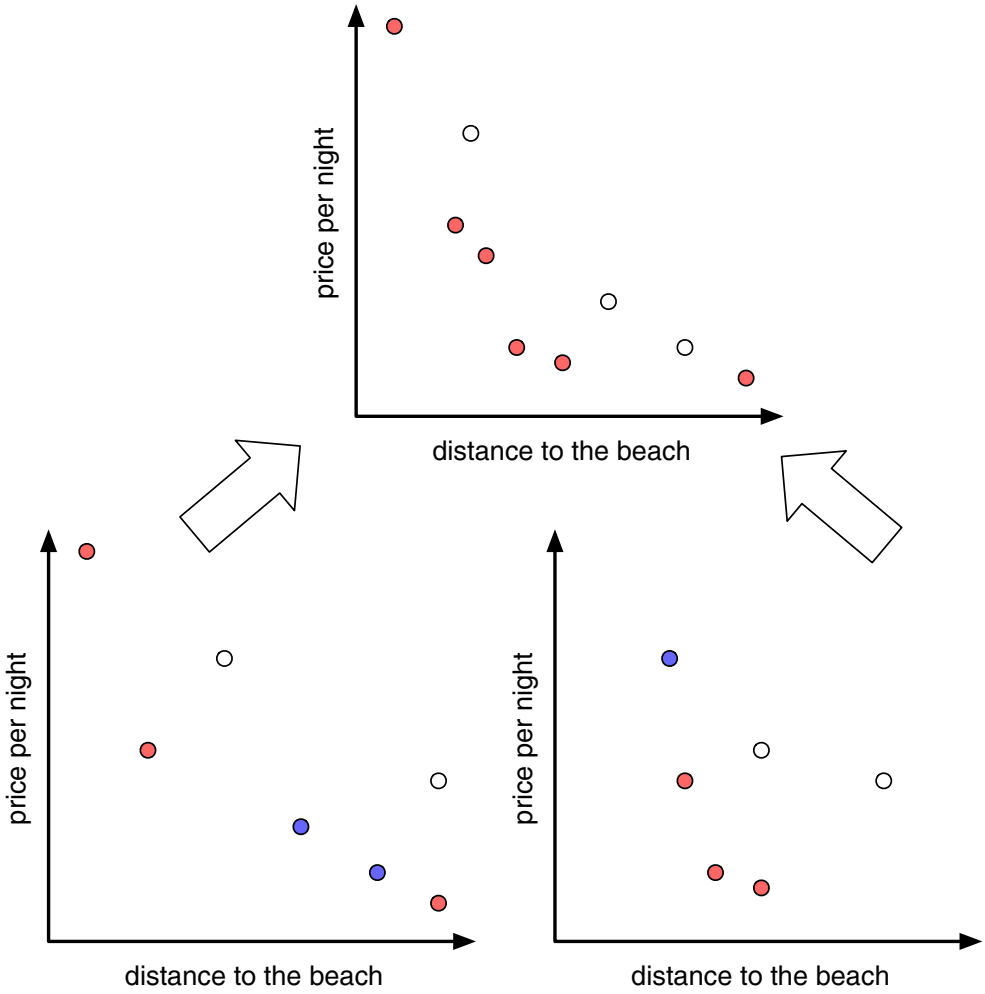

Fig. 4 Additivity of Skyline Computation

Figure 3 illustrates the concept of dominance; all data items contained in the gray shaded area are dominated by the data item in the bottom left corner – note that the dominating data item is located at the bottom left corner in our example because MIN is used for both dimensions. Intuitively, all hotels contained in the dominated region are worse than the hotel in the bottom left corner because they are more expensive or further away from the beach – most of them are both and clearly irrelevant to the user. There is one hotel with the same distance to the beach as the hotel in the skyline but it is more expensive. Hence, the hotel in the skyline is still the better choice because it is as good as or better in all dimensions (price and distance to the beach) and better in at least one dimension (price).

Another important characteristic is the *additivity* of the skyline operator, which results from the *transitivity* of the dominance relation, i.e., if a data item a dominates an item b and b dominates c, then a also dominates c. This characteristic is the basis for many efficient algorithms for computing skylines as it allows computing partial results and combining them later without having the check against the complete dataset. Given several datasets, the same result can be obtained by either (i) computing the skyline over the union of the datasets or (ii) computing the skyline first over each dataset in separate and then once more over the union of the initial skylines. The latter is illustrated in Fig. 4: the lower part shows two datasets and their skylines (the data items in the skyline are highlighted in red and blue). In the next step, the data items in the computed skylines are combined and the skyline is evaluated over the union of these two initial skylines (upper part of Fig. 4). Note that the data items highlighted in blue were only part of the initial skylines and were removed in the final step because they were dominated by data items in the other dataset. Still, all data items that have been pruned by the data items highlighted in blue could safely be pruned because of the transitivity of the dominance relation. The final skyline is the same as in Fig. 1, where the skyline has been computed over the union of the two initial datasets.

The result of a skyline query is most useful if the result set is relatively small. The size of the result set, however, strongly depends on the data distribution of the input. The

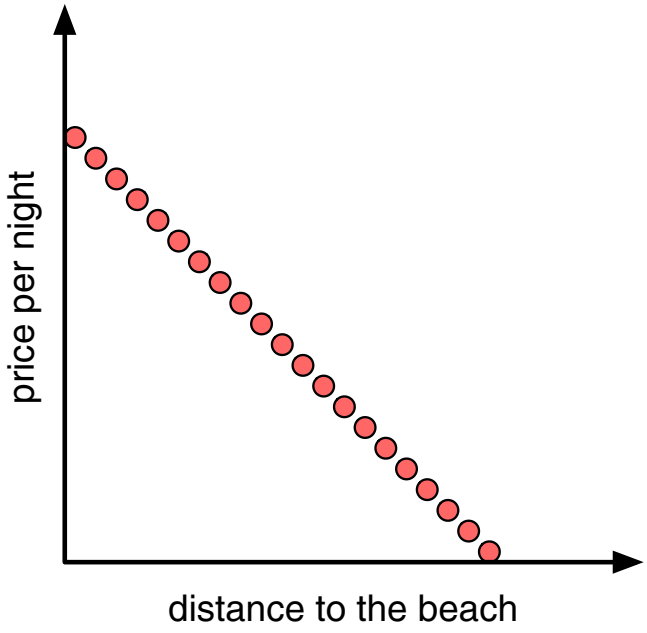

Fig. 5 Skyline over an Anti-Correlated Dataset

most problematic type of data for skyline queries is an anti-correlated dataset, i.e., if a data item is better in one dimension it is worse in the other so that (in the worst case) all data items are part of the skyline because no data item dominates another one. Figure 5 illustrates an example of such an extreme dataset and its skyline. Because of the big result size, such datasets are not only expensive to compute but the result is also less useful to the user.

A similar effect is caused by the number of dimensions, i.e., the more dimensions a skyline query is defined on, the more data items are in the skyline and the less useful the skyline is to the user. The reason for this effect is that a space with many dimensions is usually sparse. As there are only a few data items distributed in a relatively big space, it is more difficult to find data items that dominate each other, which results in a big result set.

3 Types of Skyline Queries

Throughout the years, many different types of skyline queries have been proposed. One of them is the *subspace skyline*. Such a skyline is not defined on all available dimensions (attributes of a data item) but only on a subset of them. The difference sounds very subtle but has a huge impact on computing skylines as the skyline on a subset of dimensions might contain totally different data items than the skyline on the full set of dimensions.

A skyline can be defined on derived and non-derived dimensions. A derived dimension implies that some computation is necessary to the attributes of a data item whereas for a non-derived dimension the values used for the skyline are naturally given. A skyline defined on derived dimensions is often referred to as a *dynamic skyline*. In our hotel example, both distance to the beach and price are non-derived dimensions as these values are given as attributes for each data item. However, if we assume that the user is not interested in the distance to the beach but in the distance to her current location, then the distances have to be computed on-the-fly and the resulting skyline represents a dynamic skyline.

Another way in which skylines differ is whether the user defines additional constraints. A user might, for instance, only be interested in hotels with a rating of 3 to 5 stars. Hence, the skyline should only be computed over the subset of the available data items that fulfill the user's constraints. This is commonly referred to as a *constrained skyline*.

In dependence on the available data and the application scenario, a user might wish to reduce the number of data items that she is being displayed as the result of a skyline query. Hence, *approximate skylines* have been proposed. The basic principle is to display data items representing a group of similar data items instead of showing or computing all of them. On the other hand, there are skyline query variations that display additional data items to the user. For instance, *k-skyband* queries compute all data items that are dominated by at most k data items.

Another very interesting type of skylines are *reverse skylines*. They are based on the concept of dynamic skylines but consider the problem the other way round. Instead of helping users find interesting data items, reverse skylines help companies find interested users. Assuming that the data items represent the preferences of users, then a company might define a hypothetical query object representing a potential offer. The reverse skyline then contains all user data items whose dynamic skylines contain the query object. In our hotel example, this could be used for a market analysis to find out how many users might be interested in a new offer. If there are too few interested users, the price of the hotel might be lowered to become more attractive to a broader range of potential guests.

The literature proposes many additional types of skyline queries and adapted versions of dominance to accommodate the special characteristics of different types of data, environments, and applications. Skylines have for instance been considered in the context of sensor data streams, incomplete and uncertain data, spatial data, etc.

4 Main Principles of Processing Skyline Queries

The most intuitive and basic way to compute a skyline is to compare each possible pair of data items to each other and check for mutual dominance. This naive algorithm

works but is obviously very expensive. But it can be optimized, e.g., by maintaining a window of current skyline data items in main memory and considering the data items sequentially. When a data item p is read, it is checked for dominance against each data item in the current window. If p is dominated by any data item in the window, p is pruned and not considered any further. Otherwise, if p is not dominated, it is added to the window. If p dominates data items in the current window, the dominated data items are removed. Because of the pruning, this Block-Nested-Loops algorithm is more efficient. Extensions are necessary if the window is too small to host all necessary data items and further optimization is possible in combination with sorting.

Another basic processing technique makes use of the divide and conquer principle and exploits the transitivity of the dominance relation. At first, the dataset is divided into partitions, e.g., by computing the (approximate) median in the input dataset. Then, the skyline is computed by recursively applying the partitioning until there are only a few data items per partition, which are then checked for mutual dominance. The final skyline is then computed by merging the partial skylines. Throughout the years different alternatives to define these subsets as well as other optimizations have been proposed.

Indexes, e.g., B-trees and R-trees, can also be used to speed up skyline query computation. An R-tree, for instance, can be used to determine the nearest neighbor to the best possible data item (the origin in the two-dimensional space in our hotel example). The nearest neighbor is guaranteed to be a part of the final skyline as there cannot exist any data item that is better [4, 6]. Hence, all data items contained in the region that is dominated by the nearest neighbor cannot be part of the final skyline and do not have to be processed. The final skyline can be computed by iteratively computing nearest neighbors in the non-dominated regions. As soon as the nearest neighbors are found, they can be output to the user because they are guaranteed to be part of the final skyline.

5 Recent Trends

Since their introduction in 2001, many different variants of skylines have been proposed, see [2, 3] for recent surveys. Most of them have been developed for centralized environments, where the complete dataset is available at and processed by a single machine. But skyline query computation has also been studied in distributed environments, such as P2P systems and cluster architectures, where efficient algorithms have to consider the additional communication overhead. In the context of cloud platforms, algorithms for efficient computation of skylines using MapReduce have been proposed. Moreover, skyline query computation has also been studied in the context of modern hardware, such as GPUs and FPGAs. The basic principle for all these algorithms is to exploit the types of computations that these hardware architectures are good at to speed up the computation of skylines.

The initial skyline paper [1] has inspired researchers to propose not only numerous types and variations of skyline queries but also to apply them in very different application scenarios and combine them with recent trends, e.g., skyline computation using crowdsourcing. It is therefore very likely that we will continue seeing novel research involving skylines in the future.

References

1. Börzsöny S, Kossmann D, Stocker K (2001) The Skyline Operator. In: ICDE'01, pp 421–430
2. Chomicki J, Ciaccia P, Meneghetti N (2013) Skyline Queries, Front and Back. Sigmod Rec 42(3):6–18
3. Hose K, Vlachou A (2012) A survey of skyline processing in highly distributed environments. Vldb J 21(3):359–384
4. Kossmann D, Ramsak F, Rost S (2002) Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. In: VLDB'02, pp 275–286
5. Kung HT, Luccio F, Preparata FP (1975) On Finding the Maxima of a Set of Vectors. J Acm 22(4):469–476
6. Papadias D, Tao Y, Fu G, Seeger B (2005) Progressive skyline computation in database systems. Acm Trans Database Syst (tods) 30(1):41–82