SKYLINE QUERIES FOR MULTI-CRITERIA DECISION SUPPORT SYSTEMS

by

SATYAVEER GOUD GUDALA

B.E., Osmania University, 2009

A REPORT

submitted in partial fulfillment of the requirements for the degree

MASTER OF SCIENCE

Department of Computing and Information Sciences
College of Engineering

KANSAS STATE UNIVERSITY
Manhattan, Kansas

2012

Approved by:

Major Professor
William H. Hsu

Abstract

In decision-making applications, the *Skyline query* is used to find a set of non-dominated data points (called *Skyline points*) in a multi-dimensional dataset. A data point dominates another data point if it is at least as good as the other data point in all dimensions and better in at least one dimension. The skyline consists of data points not dominated by any other data point. Computing the skyline points of a dataset is essential for applications that involve multi-criteria decision making. Skyline queries filter out the interesting tuples from a potentially large dataset. No matter how we weigh our preferences along the attributes, only those tuples which score best under a monotone scoring function are part of the skyline. In other words, the skyline does not contain tuples which are nobody's favorite. With a growing number of real-world applications involving multi-criteria decision making over multiple dimensions, skyline queries can be used to answer those problems accurately and efficiently.

This report mainly focuses on various skyline computing algorithms which can be used for online processing efficiently and are suitable to present multi-criteria decision making scenario. I implemented the Branch-and-Bound skyline Algorithm on two different data sets; one is a synthetic dataset and the other is a real dataset. My aim is to explore various subspaces of a given dataset and compute skylines over them, especially those subspace skylines which contain the least number of the skyline points.

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my major professor Dr. William H. Hsu for his guidance and support in completing my project successfully. I am grateful to him for his suggestions and comments. I also thank Dr. Mitchell Neilsen and Dr. Daniel Andresen for graciously agreeing to be on my committee.

Chapter 1 - Introduction

In computer science, algorithms for computing the Pareto frontier of a finite set of alternatives have been studied as *Skyline Query* or the maximum vector problem [Kung et al., 1975]. Given a set of choices and a way of valuing them, the **Pareto frontier** or **Pareto set** or **Pareto front** is the set of choices that are Pareto efficient. In fact, Pareto front consists of those choices that are most interesting, there by presenting the user with a few choices to choose from. In database systems the skyline query is used for supporting multi-criteria decision making applications. Given a set of d-dimensional data points, the skyline consists of those points, called skyline points, which are not dominated by any other data points. A data point dominates another data point if it is at least as good as the other data point in all dimensions and better in at least one. It was [Borzsonyi et al., 2001] who used the term skyline and also proposed a skyline extension of the SQL syntax with this form:

```
SELECT ... FROM ... WHERE ...
GROUP BY ... HAVING ...
SKYLINE OF [DISTINCT] d1 [MIN|MAX|DIFF], ..., dm [MIN|MAX|DIFF]
ORDER BY ...
```

where d_1, …, d_m are the dimensions of the skyline. For example, if a user wants to buy a car, various criteria or dimensions that he may consider may be something like the maintenance of the car, cost of the car, number of persons that can fit in the car etc.

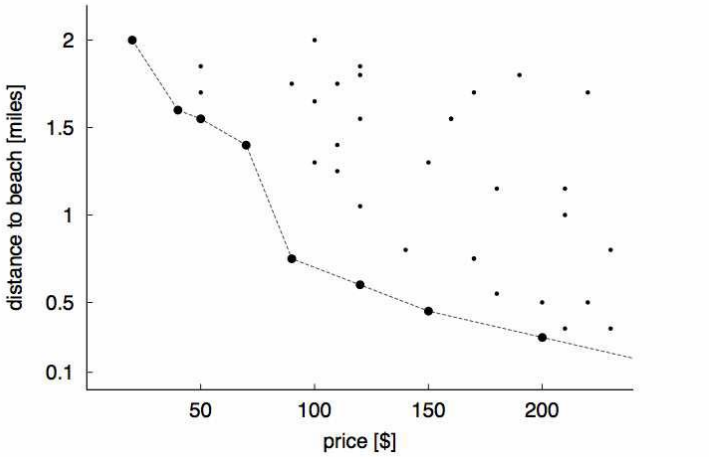

Figure 1.1 Skyline of "Cheap hotels near to the beach"

1.1 Problem Definition

Recently, the skyline operator and its computation have attracted much attention. This is mainly due to the importance of skyline results in many applications, such as multi-criteria decision making, data mining and visualization, and user-preference queries. A skyline query over d dimensions selects the points that are not dominated by any other points restricted to those dimensions. Unfortunately, as the number of dimensions increases, the chance of one point dominating another point is very low. As such, the number of skyline points become too numerous to offer any interesting insights. To find more important and meaningful skyline points in high dimensional space we need to explore subspaces and calculate skylines over the subspaces to present the user with a less number of candidates.

	Price	*Dist.*	*Rating*
H_1	4	3	2
H_2	5	1	1
H_3	1	4	4
H_4	3	5	5
H_5	2	2	3

Table 1.1 Example of hotel dataset (synthetic data)

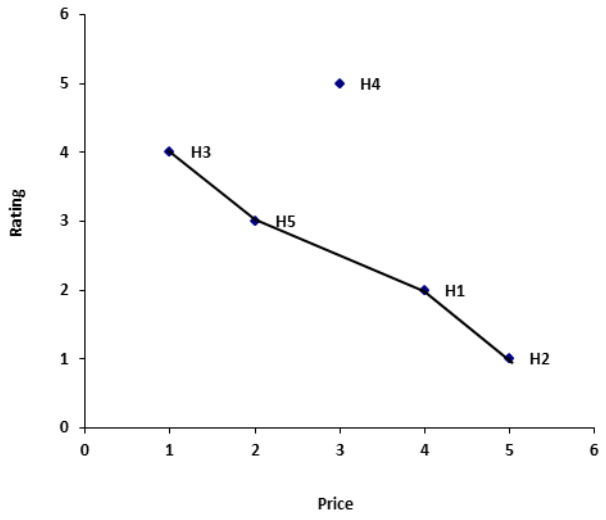

Figure 1.2 Skyline on price and rating

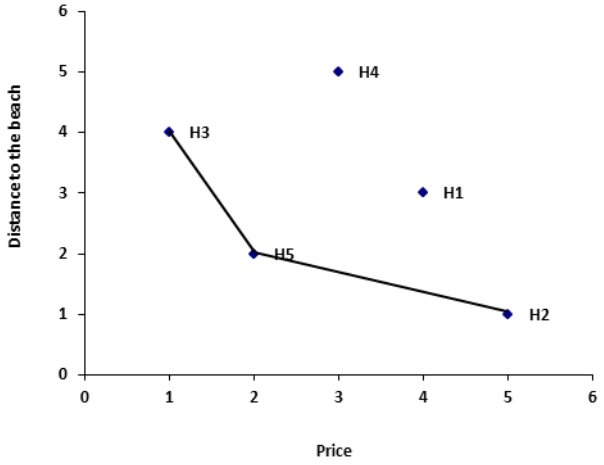

Figure 1.3 Skyline on price and dist.

Consider a typical skyline query example as follows: a travel agency has a list of hotels online, each with price, dist. (distance to beach), and rating as attributes. Assume there are five hotels as listed in Table 1-1. A skyline query issued on price and rating attributes will give the result $\{H_3, H_5, H_1, H_2\}$ as skyline points as shown in Figure 1.2. Whereas a skyline query issued with a combination of price and dist. will give $\{H_3, H_5, H_2\}$ as skyline points. So, exploring skylines of all possible non-empty subsets of a given set of dimensions especially those which contain least number of skyline points helps in presenting a user with a manageable number of interesting points to choose from.

Input

A set of points $p_1, p_2, \ldots . p_n$ each with *'d'* dimensions.

Output

A set of points P (referred to as the skyline points), such that any point $p_i \in P$ is not dominated by any other point in the dataset.

Objective

Minimize the number of skyline points by exploring various subspaces of the dataset.

Chapter 2 - Review of Literature

2.1 Why Multi-Criteria?

In our everyday-life, people consider multiple criteria in decision making. For example, when a person wants to buy a car, the various criteria for decision making are the cost of the car, the maintenance of the car, the capacity of the car, i.e., the number of persons and the mileage etc. Consider the following figure; it is 'Consumer Reports' magazine for best and worst cars. It uses four different criteria for evaluating the cars namely Reliability, Fuel Economy, Performance and Safety. Applying skyline queries over those criteria will give us cars which are at least as good as the other cars in all criteria and better in at least one criterion. So, incorporating multi-criteria in decision making helps to present user with more quality results.

Figure 2.1 Consumer reports magazine for cars.

Following gives few more examples as to why we should consider multi-criteria. A movie is categorized in several genres like family, action, horror, comedy, suspense. Hence single rating schema can't sufficiently explain the quality of any product. For example *Titanic* can't be compared with *Evil Dead* though they are top class movies. Many restaurant guides provide four criteria for restaurant ratings: food, decor, service and cost. Customer preference for

specific vacation packages can depend significantly on the time of the year or other temporal considerations. In the dynamic web content application, customer wants to see different type of information during different types of the day. Frequently there is no clear winner over all criteria, i.e. a high-priced product with high quality and a cheap product with lower quality. Surely we are not interested in a product with a high price and poor quality; products of this kind are dominated by another one, as there are others which fulfill our criteria better. The products which are not dominated are more or less equally good based on our given criteria, and we need to apply additional criteria to make a decision. Hence, the skyline queries are used to find out these non-dominating solutions. A skyline query returns all non-dominated solutions, and also we have variations of skyline queries which return the most interesting points in the data space defined by the constraints.

2.2 Challenges in Incorporating Multiple-criteria

Conflicting criteria is one of the challenges in incorporating multi-criteria. Different criteria represent different dimensions of the alternatives, there is always a possibility that they may conflict. When we consider the problem of choosing a restaurant, we may have conflicting criteria: for example, the food of a restaurant is delicious while the decor is primitive. Another problem in multi-criteria decision making is that there will be no overwhelmingly good item i.e. an item which is best on all dimensions. Incommensurable units are also a challenge in multi-criteria decision making. This is because different criteria may be associated with different units For example consider the case of buying a car, the criteria “cost” and “mileage” will be measured different units like dollars and MPG. So, it is this problem of having to consider different units, makes multi-criteria decision making more complex. Likewise, there are few more challenges in multi-criteria decision making, like in the prediction phase the problem of estimating unknown ratings in a multi-dimensional recommendation space, demand for on-the fly estimation, and the last being application of our methods to real life personalization problems and also experimental validation of their performance.

2.3 Different Approaches for Multi-criteria Decision Making

	Type of information from the decision maker	**Salient feature of information**	**Major classes of the methods**
Multi-Criteria Decision Making	No Information		Dominance Maximin Maximax
		Standard Level	Conjunctive method Disjunctive Method
	Information on attributes	Ordinal	Elimination by aspect Lexicographic semi order Lexicographic method
		Cardinal	Weighted sum model Weighted product model Analytic Hierarchy Process ELECTRE TOPSIS

Table 2.1 A Taxonomy of MCDM methods (according to Cheng and Hwang [1991])

2.3.1 Weight-Based Approach

The weight-based approach is the commonly used approach, especially in single dimensional problems. Different weights for all criteria have to be provided by the user to reflect the relative importance when making decisions among all the alternatives. Takes a linear weighted combination of multiple criteria and reduce problem to a single-criterion optimization. If there are m alternatives and n criteria then, the best alternative is the one that satisfies (in maximization case) the following expression

$$A_{WSM} = max_i \sum_{j=1}^{n} a_{ij} w_j \quad \text{for i} = 1, 2, 3, \ldots, \text{m}$$

Where $A_{WSM\text{-}Score}$ is the Weighted Sum Score (WSM) of the best alternative, n is the number of the decision criteria a_{ij} is the actual value of the *i-th* alternative in terms of the *j-th* criterion, and W_j is the weight of importance of the *j-th* criterion.

Limitations

In single-dimensional cases WSM method can be applied without any issues but in case of multi-dimensional MCDM problems, applying WSM becomes difficult. And also weights of different criteria vary with different user and time.

2.3.2 Analytic Hierarchical process

AHP is a process for developing a numerical score to rank each decision alternative based on how well each alternative meets the decision maker's criteria. In AHP, problems are decomposed into hierarchy of criteria and alternatives. A matrix of criterion preferences from user is taken and weights of each criterion are calculated from the algorithm.

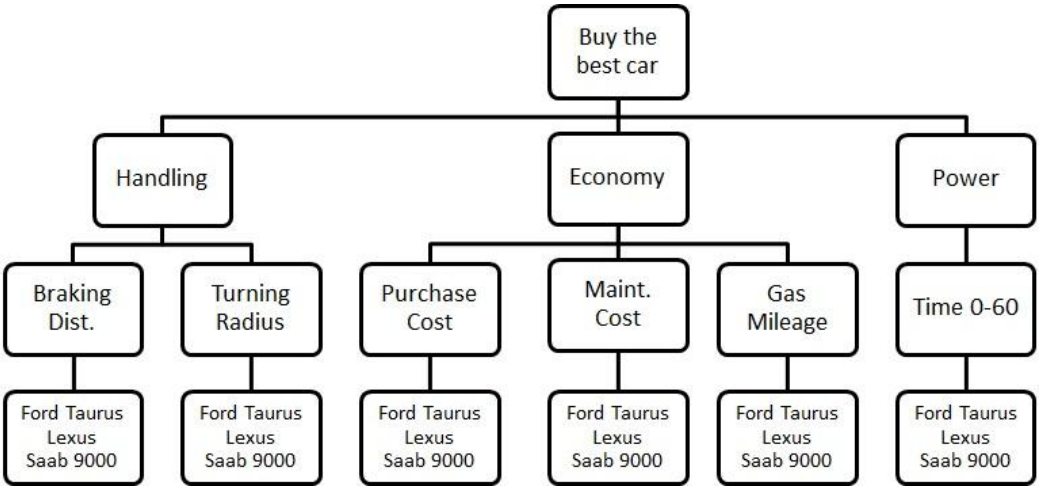

Figure 2.2 AHP hierarchy for buying a car

When we want to compare the best car in terms of Economy, we take the input from the user and the relative criteria. In our example, we shall consider three attributes Purchase Cost (P), Main Cost (M) and Gas Mileage (G).

	P	M	G
P	1	3	5
M	1/3	1	3
G	1/5	1/3	1

Table 2.2 Matrix of Criterion Preferences

For example, here matrix indicates that P is 5 times more important that of G. Weights are calculated by normalizing the columns.

$W_P = \frac{15}{23} = 0.65$, $W_M = \frac{5}{23} = 0.22$, $W_G = \frac{3}{23} = 0.13$

Through obtained weights, ratings to the item are predicted.

2.3.3 Skyline Queries

Skyline query is one of the most widely used preference queries. It is based on the dominance relationship between tuples (Leewuen, 2010). Skyline query filters out set of all interesting point in the dataset. A point is said to be interesting if it is not dominated by any other point. A point dominates another point if it is good or better in all dimensions and better in at least one dimension. It is based on the dominance relationship between tuples. Assuming that smaller values are preferable in all dimensions, the dominance relationship and the skyline is defined as follows (Leewuen, 2010)

Definition 1 (dominant tuple): A tuple t_i dominates another tuple t_j ($t_i < t_j$), if and only if t_i is smaller than or equal to t_j in all dimensions and it is strictly smaller than t_j in at least one of them.

Definition 2 (Skyline): The skyline consists of the tuples not dominated by any other tuple.

Example

The following dataset taken from (Sankaranarayanan) clearly explains the concept skyline Queries. The dataset contains information about hotels, their distance to the beach and the price. A two dimensional plot of the dataset is shown in Figure 2.3; the distance and price are assigned to the X, Y axis of the plot respectively. The goal is to find a hotel whose distance to beach and the price are both minimum. The dataset may not have one single point that satisfies both desirable properties. The user is presented with a set of interesting points that partly satisfies the imposed constraints. The interesting points in the dataset will be:

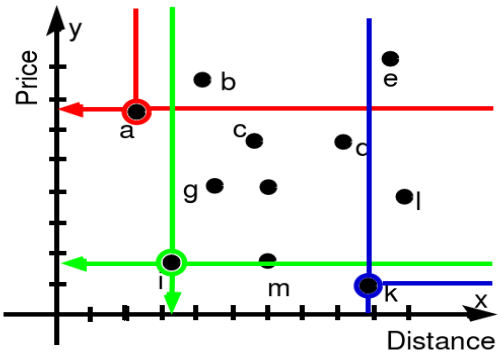

Figure 2.3 Hotel data set (Sankaranarayanan)

'a' has the shortest distance from the beach

'k' has the least price

'i' has neither shortest distance nor the least price

All the other points are dominated by set of points {a, i, k}, i.e., both distance and price values are greater than one or more skyline points.

Chapter 3 - Existing Skyline Algorithms and Related Work

3.1 Translating Skyline query into nested SQL

Borzsonyi *et al*., [2001] show how skyline queries can be executed on top of a relational database system. The following shows the corresponding translation of the skyline query into nested SQL.

3.1.1 Skyline query in SQL (for hotel dataset)

```
SELECT *
FROM Hotels h
WHERE NOT EXISTS(
     SELECT *
     FROM Hotels h1
     WHERE h1.distance <= h.distance AND
           h1.price <= h.price AND
          (h1.distance < h.distance OR h1.price < h.price));
```

But the above approach gives very poor performance for following reasons:

- It corresponds to the naive "nested-loops" way to compute the skyline because this query cannot be un-nested.
- If the skyline query involves a join or group-by, this join or group-by would have to be executed as part of the outer query and as part of the sub query.

3.2 Algorithms for computing Skyline Points

Since the introduction of the skyline operator [Borzsonyi *et al*., 2001], a number of secondary memory algorithms have been developed for efficient skyline computation. These algorithms can be classified into two categories. The first one involves solutions that do not require any preprocessing on the underlying dataset. Algorithms such as Block Nested Loops (BNL) [Borzsonyi *et al*., 2001], Divide and Conquer (D&C) [Borzsonyi *et al*., 2001], Sort First Skyline (SFS) [Chomicki *et al*., 2003], and Linear Elimination Sort for Skyline (LESS) [Godfrey *et al*., 2005] belong to this category. The algorithms in the second category utilize different index structures such as sorted lists and R-trees to reduce the query costs. Well-known algorithms in this category include Bitmap [Tan *et al*., 2001], Index [Tan *et al*., 2001], Nearest Neighbor (NN) [Kossmann *et al*., 2002], and Branch-and-Bound (BBS) [Papadias *et al*., 2003, 2005].

The advantage of index-based algorithms is that we need to access only a portion of the dataset to compute the skyline, while non-index-based algorithms have to visit the whole dataset at least once. However, index-based algorithms have to incur additional time and space costs for building and maintaining the indexes.

3.2.1 Block Nested Loop

[Borzsonyi *et al*., 2001] proposed a naive way to compute skyline. A straightforward approach to compute the skyline is to compare each point p with every other point; if p is not dominated, then it is a part of the skyline. BNL builds on this concept by scanning the data file and keeping a list of candidate skyline points in main memory. The first data point is inserted into the list. For each subsequent point p, there are three cases: i) If p is dominated by any point in the list, it is discarded as it is not part of the skyline. ii) If p dominates any point in the list, it is inserted into the list, and all points in the list dominated by p are dropped. iii) If p is neither dominated, nor dominates, any point in the list, it is inserted into the list as it may be part of the skyline.

The list is self-organizing because every point found dominating other points is moved to the top. This reduces the number of comparisons as points that dominate multiple other points are likely to be checked first. A problem of BNL is that the list may become larger than the main memory. When this happens, all points falling in third case (cases (i) and (ii) do not increase the list size), are added to a temporary file. This fact necessitates multiple passes of BNL. In particular, after the algorithm finishes scanning the data file, only points that were inserted in the list before the creation of the temporary file are guaranteed to be in the skyline and are output. The remaining points must be compared against the ones in the temporary file. Thus, BNL has to be executed again, this time using the temporary (instead of the data) file as input. The advantage of BNL is its wide applicability, since it can be used for any dimensionality without indexing or sorting the data file. Its main problems are the reliance on main memory (a small memory may lead to numerous iterations) and its inadequacy for on-line processing (it has to read the entire data file before it returns the first skyline point).

Complexity: The best case complexity is O(n), and that is when all the candidate skyline points fit in the memory at every instance. It's obvious, that in the worst case, the complexity is $O(n^2)$ which is the same as the naive nested loop algorithm.

3.2.2 Divide and Conquer

The D&C approach divides the dataset into several partitions so that each partition fits in memory. Then, the partial skyline of the points in every partition is computed using a main-memory algorithm and the final skyline is obtained by merging the partial ones. In order to obtain the final skyline, we need to remove those points that are dominated by some point in other partitions. For large datasets, the partitioning process requires reading and writing the entire dataset at least once, thus incurring significant IO cost. Further, this approach is not suitable for on-line processing because it cannot report any skyline until the partitioning phase completes.

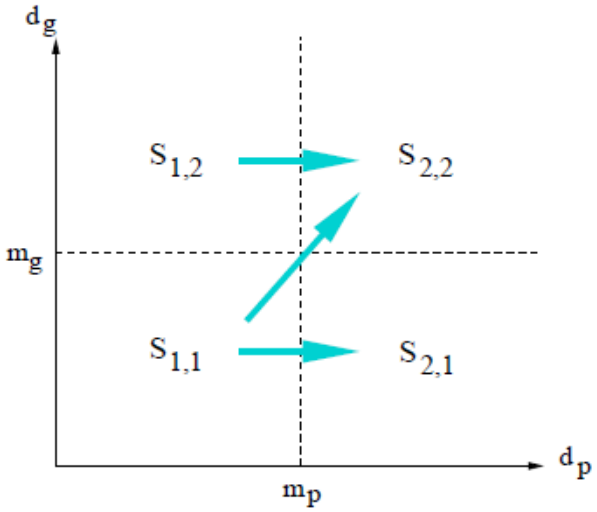

Figure 3.1 Divide and Conquer (Borzsony, Kossamn, & Stocker, 2001)

Complexity: Both the best case and the worst case complexity is O(n (log n)d.2) + O(n log n); where n is the number of input tuples and d is the number of dimensions in the skyline. Hence, we expect this algorithm to outperform BNL in worst cases and to be worse in good ones.

3.3 Motivation for an on-line algorithm

1. The above algorithms are not suitable for interactive applications like Recommendation Systems (which demand the results to be displayed almost immediately) because these algorithms require reading the whole dataset at least once before the first results are returned.
2. A skyline may consist of thousands of points and it is less important that all points of the skyline are produced: it is sufficient to give the user a big picture. In a mobile

environment, as the user moves on, this big picture of interesting points (e.g., restaurants) must be recomputed continuously, making it even more important to compute a big picture fast rather than a complete result. These observations motivate the need for an online algorithm to compute the skyline.

3. The existing algorithms to compute the skyline work in a batch-oriented way and these algorithms involve reading the whole data set and return the first results (almost) at the end of their running time.
4. In addition, the running time of these algorithms can be very high and there is little hope to find better batch-oriented algorithms: it can be shown that the best algorithm to compute the full skyline has a (worst-case) complexity of $O(n(\log n)^{d-2})$, with n the number of points in the data set and d the number of dimensions; this is a higher complexity than sorting.
5. An online algorithm would probably take much longer than a batch-oriented algorithm to produce the full skyline, but an online algorithm would produce a subset of the skyline very quickly

3.4 The following properties should be satisfied by an online algorithm

Kossmann *et al*., [2002] proposed a set of criteria for evaluating online algorithms:

1. The first results should be returned almost instantaneously.
2. The algorithm should produce more and more results the longer the algorithm runs. If given enough time, the algorithm should produce the full skyline.
3. The algorithm should only return points which are part of the skyline. In other words, the algorithm should not return good points (e.g., good restaurants) at the beginning and then replace these good restaurants with better restaurants.
4. The algorithm should be fair. In other words, the algorithm should not favor points that are particularly good in one dimension; instead it should continuously compute skyline points from the whole range.
5. The user should have control over the process. In other words, it should be possible to make preferences while the algorithm is running. Using a graphical user interface, the user should be able to click on the screen and the algorithm will return next points of the skyline which are near the point that the user has clicked on.

6. The algorithm should be universal with respect to the type of skyline queries and data sets. It should also be based on standard technology (i.e., indexes), and it should be easy to integrate the algorithm into an existing database system. For a given data set (e.g., hotels or restaurants) one index should be enough to consider all dimensions that a user might find interesting.

3.5 Branch-and-Bound Skyline Algorithm (BBS)

The Branch-and-Bound Skyline algorithm proposed by [Papadias *et al.*, 2003, 2005] satisfies the properties demanded for an on-line algorithm. It is based on Nearest Neighbor Search and R-tree indexing.

3.5.1 R-tree Indexing

R-trees are tree data structures that are similar to B-trees, but are used for spatial access methods i.e. for indexing multi-dimensional information. The data structure splits space with hierarchically nested, and possibly overlapping, minimum bounding rectangles (MBRs, otherwise known as bounding boxes, i.e. "rectangle", what the "R" in R-tree stands for). Each node of an R-tree has a variable number of entries. Each entry within a non-leaf node stores two pieces of data: a way of identifying a child node, and the bounding box of all entries within this child node. Nodes correspond to disk pages if the index is disk-resident, and the structure is designed so that a spatial search requires visiting only a small number of nodes The index is completely dynamic; inserts and deletes can be inter-mixed with searches and no periodic reorganization is required.

Let ‘M’ be the maximum number of entries that will fit in one node and let $m \leq M/2$ be a parameter specifying the minimum number of entries in a node An R-tree satisfies the following properties:

1. Every leaf node contains ‘between m and M index records unless it is the root
2. For each index record (I, tuple-identifier) in a leaf node, I is the smallest rectangle that spatially contains the n-dimensional data object represented by the indicated tuple
3. Every non-leaf node has between m and M children unless it is the root.
4. For each entry (I, child-pointer) in a non-leaf node, I is the smallest rectangle that spatially contains the rectangles in the child node.

5. The root node has at least two children unless it is a leaf.
6. All leaves appear on the same level.

3.5.2 Description of BBS

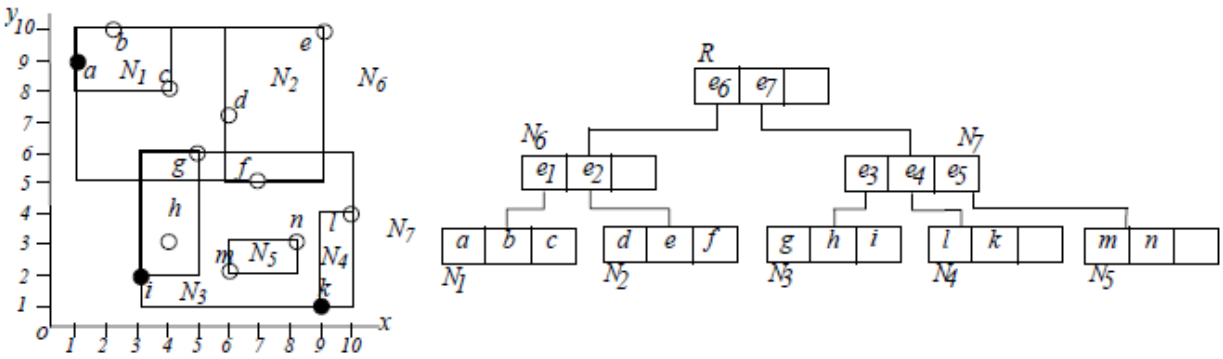

Figure 3.2 Spatial representation of 2d points and corresponding R-tree (Papadias, Fu, JP Morgan Chase, & Seeger, 2005)

BBS is based on nearest neighbor search and uses R-tree indexing. The algorithm consists of the following steps.

Index: First we have to use an R-tree to index the objects (which is a pre-requisite for BBS) and set the list of skyline points (S) to Null.

Access Root: Insert all the entries of the root in a heap, and sort them according to their minimum distance from a minimum point in ascending order. For positive values this point could be the origin while for negative values, this point could be a point that consists of the minimum values at each direction. The minimum distance (*mindist*) of a node of the R-tree to a point is given by the Euclidean distance of the point to the lower left corner of the node's Minimum Bounding Rectangle (MBR). The *mindist* of two points is their Euclidean distance.

Compute Skyline Points: The following gives the steps to compute skyline points:

– Expand the entry *e* in the root which has the lowest *mindist*,

– Compare e with all the entries in S and if *e* is "dominated" by any point in S, discard *e*

– If *e* is not "dominated" and *e* is an intermediate node, insert each child of e that is not dominated, in the heap

– If *e* is a data point, insert it in the skyline list S.

Complexity: The main memory requirement for the BBS is at the same order of the size of skyline list, since both the heap and the main-memory R-tree sizes are of at this order.

Furthermore, the number of node accesses by BBS is at most ***s.h***, where s is the number of skyline points, and h the height of the R-tree.

Action	Heap Contents	S
Access root	$<e_7, 4><e_6, 6>$	Ø
Expand e_7	$<e_3, 5><e_6, 6><e_5, 8><e_4, 10>$	Ø
Expand e_3	$<\boldsymbol{i, 5}><e_6, 6><h, 7><e_5, 8> <e_4, 10><g, 11>$	$\{i\}$
Expand e_6	$<h, 7><e_5, 8><e_1, 9><e_4, 10><g, 11>$	$\{i\}$
Expand e_1	$<\boldsymbol{a, 10}><e_4, 10><g, 11><b, 12><c, 12>$	$\{i, a\}$
Expand e_4	$<\boldsymbol{k, 10}> < g, 11>< b, 12>< c, 12>< l, 14>$	$\{i, a, k\}$

Table 3.1 Skyline computation over R-tree example shown in Figure 3.2

3.5.3 Pseudo code of BBS Algorithm

The following is the pseudo code for the Branch-and-Bound Skyline Algorithm proposed by (Papadias, Fu, JP Morgan Chase, & Seeger, 2005):

Algorithm BBS (R-tree *R*)
S=∅ // *list of Skyline points*
insert all entries of the root *R* in the heap
while heap not empty
remove top entry *e*
if *e* is dominated by some point in *S* discard *e*
else // *e is not dominated*
if *e* is an intermediate entry
for each child e_i of *e*
if e_i is not dominated by some point in *S*
. insert e_i into heap
else // *e is a data point*
insert e_i into *S*
end **while**
End BBS

3.6 Variations of Skyline Queries

Ranked Skyline Queries: In addition to the distinct score functions used for skyline computation a "total" score function is provided. Then all skyline points are returned in order of the "total" score function. An alternate preference function is used instead of the minimum criterion. The priority queue uses the alternate preference-function to compute Minimum Distance to the elements in the queue.

Constrained Skyline Queries: In addition to the usual skyline input parameters, the data space is constrained and the skyline queries return skyline points only from the constrained data-space. This is typically done by specifying a hyper-rectangle in which all data items have to be located. When inserting objects in to the priority queue, objects that lie completely outside the constraint region are pruned.

Enumerating Queries: Enumerating queries return, for each skyline point *p*, the number of points dominated by *p*. This is done by defining the spatial bound for the region where a skyline point dominates. Then all points in the datasets are scanned and checked against the spatial extent for each of the skyline points. The total number of point-region intersections gives the required count for each skyline points.

K-dominant Queries: K-dominating queries return those 'K' points that dominate the largest number of other points. In fact this is not a skyline query because the result does not necessarily contain the skyline points.

Chapter 4 - Implementation and Results

The algorithm that I've implemented is the Branch-and-Bound Skyline (BBS). The implementation of BBS consists of two steps. In the first step, we need to preprocess the dataset i.e. create an R-tree index for the dataset. In the second step, load the stored index and find the Skyline points by applying the algorithm. For indexing the dataset, I extended the Spatial Index Library by Marios Hadjieleftheriou (Hadjieleftheriou, 2011). First, I've implemented on *Car Evaluation* dataset available at University of California, Irvine's machine learning repository. The Car Evaluation database contains examples that relate Car to the six input attributes: buying, maintenance, doors, persons, lug boot, safety. The following gives the list of attributes and their corresponding values.

Attribute	**Possible Values**
Buying	vhigh, high, med, low.
Maintenance	vhigh, high, med, low.
Doors	2, 3, 4, 5more.
Persons	2, 4, more.
Lug boot	small, med, big.
Safety	low, med, high.

Table 4.1 Car Evaluation dataset attribute values

The number of instances was 1728 and instances completely cover the attribute space. When I issued a skyline over all six attributes, the skyline had 1512 points, which is bad for recommendation purposes because the result had almost the entire dataset. Even if we reduce the number of attributes and issue a skyline only on one attribute (let's choose *buying*) and one-dimensional skyline is trivial, it is equivalent to computing min, max or distinct, the skyline still has 438 points because the attribute space of buying has only four choices, so, one fourth of the database will have min value. This is the main disadvantage of skyline Queries because there is no upper bound on the skyline points, in worst cases the whole database will be part of the skyline. So, skyline queries on this kind of data set are not useful for recommendation purposes.

Another data set that I've used is the NBA data set from (Databasebasketball Website); it consists of some statistics of the players in the NBA history. My aim was to find the most interesting players in NBA history. I've considered six attributes namely Total Points (PTS),

Rebounds (REB), Assists (AST), Steals (STL), Field Goals Made (FGM) and Free Throws Made (FTM) and the dataset had 3941 points (players). The following table shows skyline query results over various subsets of attributes, especially the subspace skylines which contain least number of the skyline points.

Total Points	Rebounds	Assists	Steals	Field Goals Made	Free Throws Made	Skyline Points
X	X	X	X	X	X	23
X	X		X	X	X	11
X	X			X	X	4
X			X	X	X	4
X				X	X	2
X				X		1

Table 4.2 Skyline query results on NBA dataset

4.1 Correctness

To prove the correctness of the implementation, I've used the translation of the skyline queries into the SQL proposed in [Borzsonyi *et al*., 2001]. The following gives the SQL translation of the skyline query on all the six attributes of the NBA dataset:

```
SELECT n.name
FROM NBA n
WHERE NOT EXISTS (
     SELECT n1.name
     FROM NBA n1
     WHERE n1.pts >= n.pts AND n1.reb >= n.reb AND n1.ast >=
     n.ast AND n1.stl >= n.stl AND n1.fgm >= n.fgm AND n1.ftm >=
     n.ftm AND (n1.pts > n.pts OR n1.reb > n.reb OR n1.ast >
     n.ast OR n1.stl > n.stl OR n1.fgm > n.fgm OR n1.ftm >
     n.ftm));
```

The skylines obtained by applying the algorithm and the skyline resulted from querying the database were equal. This proves the correctness of the implementation.

Chapter 5 - Applications

5.1 Key Applications

Skyline queries and its variations are used in many applications involving multi-criteria decision-making, business intelligence, data visualization and others.

Multi-criteria decision making

In decision-making applications, the skyline query can be used to find a set of dominating data points (called skyline points) in a multi-dimensional dataset. Skyline queries can be tuned according to user preferences so that we can return user-specific results.

User preference queries

The skyline operator offers a good start to provide the functionality of preference queries in relational databases, and would be easy for users to employ.

Business intelligence

Skyline queries can be used as a business intelligence tool. For example, a skyline query can be used to determine customers who buy much and complain little.

Data visualization

Through skyline, the outline of a geometric object can be determined; in other words, the points of a geometric object that are visible from a certain perspective can be determined using a skyline query.

5.2 Conclusion and Future Work

With Recommendation Systems making a significant process in all the recent e-commerce applications, skyline queries can be employed for recommendation purposes. Hence this R-tree indexing of database system plays a crucial role next generation recommendation systems, especially recommendations over very large databases. There is a lot of ongoing research on skyline queries, some of the future work in skyline queries can be to find alternative optimal and progressive algorithm for high dimensional spaces where R-trees are inefficient, and also fast retrieval of approximate skyline points i.e. points that do not belong to skyline but are very close.

References

Borzsony, S., Kossamn, D., & Stocker, K. (2001). The Skyline operator. *ICDE*, (pp. 421 - 430).

Databasebasketball. Retrieved August 2011, from Databasebasketball Website: http://databasebasketball.com/stats_download.htm

Guttman, A. (1984). *R-trees: A Dynamic Index Structure for Spatial Searching.*

Hadjieleftheriou, M. (2011). *Libspatialindex*. Retrieved August 2011, from GitHub Inc Website: http://libspatialindex.github.com/

Kossman, D., Ramsak, F., & Rost, S. (2002). Shooting Stars in the Sky: An Online Algorithm for Skyline Queries. *VLDB*, (pp. 275-286).

Leewuen, J. V. (2010). Theory and Practice of Computer Science. *SOFSEM.*

Papadias, D., Fu, G., JP Morgan Chase, & Seeger, B. (2005). Progressive Skyline Computation in Database Systems. *ACM Trans. Database Syst*.

Sankaranarayanan, J. *Skyline queries and its variations. An Optimal and Progressive Algorithm for Skyline Queries.*