Robust Cardinality and Cost Estimation for Skyline Operator

Surajit Chaudhuri* Nilesh Dalvi† Raghav Kaushik*

*Microsoft Research, Redmond. †University of Washington, Seattle.

Abstract

Incorporating the skyline operator inside the relational engine requires solving the cardinality estimation and the cost estimation problem, hitherto unaddressed. We propose robust techniques to estimate the cardinality and the computational cost of Skyline, and through an empirical comparison, show that our technique is substantially more effective than traditional approaches. Finally, we show through an implementation in Microsoft SQL Server that skyline queries can substantially benefit from our techniques.

1 Introduction

Database searches often return empty answers when not all of the user requirements are satisfied. This is the case when a user wants to search for the best flight, the perfect house or used car. In such cases, a user wants to specify a set of preferences. Traditional database engines or query languages do not have support for preferences. Recently, many extensions to the SQL language have been proposed, such as *Preference SQL* [6]. An example query with preferences looks as follows:

```
SELECT *
FROM UsedCars
WHERE make = 'Toyota'
PREFERRING mileage LOW, price LOW
```

This query represents a user looking for a used car with make `Toyota` preferring low mileage and low price. The query returns all `Toyota` cars which are not "dominated", in that there is no other `Toyota` car that has a lower mileage and a lower price.

The *Skyline* operator has been proposed [15, 16, 5] to implement preference queries. Skyline takes a set of preferences as input, and returns only those tuples for which there is no other tuple that is better with respect to all preferences.

While the Skyline operator can be expressed in SQL, it is widely recognized that an efficient implementation requires introducing an operator inside the database engine for computing the skyline [15]. Several physical implementations for the Skyline operator have been proposed [15, 16, 5] that significantly out-perform a direct implementation of skyline using SQL. However, very little attention has been focused on the other challenges associated with implementing skyline as an operator (both logical and physical) within a relational engine, namely cost and cardinality estimation.

We begin by discussing how it is not sufficient to have skyline as the top-most operator in an operator tree, and how the interaction of skyline with other relational operators can result in it being pushed down leading to significant performance benefits. This motivates the need for *robust* cardinality and cost estimation for the skyline operator. We observe properties of skyline that distinguish it from traditional relational operators such as selection. We show, for instance, that unlike selections where adding a new selection can only decrease the cardinality, adding a new preference can increase the skyline cardinality, indeed up to the size of the entire relation. We argue that these properties introduce unique challenges in estimating the cardinality and cost of the skyline operator.

We next address the cardinality estimation problem. We begin by proving a theorem on the cardinality of skyline when the data points are generated from an arbitrary distribution. This provides us with a basis for cardinality estimation in conjunction with any synopsis structure that captures this distribution, such as single- and multi-dimensional histograms, samples, wavelets, etc. Even under the attribute value independence assumption commonly used in query optimizers, skyline cardinality estimation poses substantial challenges. Prior work [1, 3] has addressed the problem of skyline cardinality estimation but only under strong assumptions in addition to attribute value independence, such as assuming that all attributes are unique and completely ordered. Since categorical attributes are very common, this assumption is severely restrictive. Our first contribution is to apply the above theorem to relax this assumption and derive cardinality estimates for categorical attributes.

Our next contribution is to relax the attribute value independence assumption. While independence assumptions

are known to lead to erroneous cardinality estimates, the problem is further exacerbated in the case of the Skyline operator because of its distinctive properties noted above. We use uniform random sampling to address correlation in the data. We show that a naive application of sampling incurs high errors, and that sampling has to be used with care for skyline cardinality estimation.

We then propose cost estimation solutions for two of the physical implementations proposed so far in prior work — Block-nested-loop Algorithm [15], and Block-nested-loop with Presorting [5]. Prior work [15] has shown that the skyline computation is CPU-intensive. Hence, we focus on estimating the CPU cost of skyline computation. The CPU cost estimation requires an estimate of the number of comparisons performed during the execution of the skyline operator. We provide a solution to this novel estimation problem. Our solution has the interesting property that any cardinality estimation algorithm can be plugged in to work with it. Hence, further refinements of our cardinality estimation solution and other approaches to skyline cardinality estimation can all be integrated into our cost estimation solution.

We study our solutions through an empirical evaluation where we vary both the data distribution and the physical implementation of the skyline. We show that our cost estimation and cardinality estimation techniques yield tolerably low errors across the spectrum. We also implement our techniques in Microsoft SQL Server and demonstrate that our techniques yield substantial benefits in terms of query answering time.

In Section 2, we define the skyline operator and review the physical implementations we focus on in this paper. We motivate the need for robust cardinality and cost estimation in Section 3. We discuss techniques for estimating Skyline cardinality in Section 4 and the cost of Skyline in Section 5. Section 6 reports experiments, Section 7 related work, and we conclude in Section 8.

2 Skyline Operator and Physical Implementation

In this section, we review the definition of the Skyline operator and previously proposed physical implementations.

2.1 Preferences

Let T be any relational table with a set of attributes. A preference is a partial order on the set of tuples. In this paper, we consider two ways to specify a preference: using a *predicate* preference and using a *numeric* preference. A predicate preference is a predicate on T, e.g. *(color='red')* and *(price < 10K)*, that defines a partial order where all tuples that satisfy the predicate come before all other tuples. A *numeric* preference is defined over a numeric attribute using an ordering of the domain, where a tuple appears before another if it has a lower value as per the ordering. An example of numeric preference is *(price LOW)*. It defines a partial order where a tuple comes before another tuple if it has smaller price.

2.2 Skyline

Let P be a set of preferences. We say that a tuple t_1 dominates another tuple t_2 with respect to P if it appears before t_2 (i.e., is "better" than t_2) with respect to at least one atomic preference, and is not below (i.e., is at least as good as) t_2 in all other preferences. Given table T, the Skyline of T with respect to preferences P, denoted as $\mathcal{S}_P(T)$, is the set of tuples in T that are *not dominated* by any other tuple.

Example 2.1 Consider a set of cars with attributes <*price, year*>. Let the tuples in this set be {<15000,1999>, <18000,2004>, <20000,2003>}. A skyline on *price LOW* would return the cheapest car, i.e., <15000,1999>. A skyline on *year HIGH* would return the newest car, i.e., <18000,2004>. On the other hand, a skyline on *price LOW, year HIGH* would return two cars, {<15000,1999>, <18000,2004>}, since each of these cars is better than the other on at least one attribute. The car <20000,1997> is not returned as it is dominated by the car <18000,2004>.

2.3 Algorithms for Computing Skyline

The problem of computing the Skyline has also been studied under the name of maximum vector problem [8]. While this work assumed that the whole set of points fit into memory, several algorithms suitable for a database system have been proposed [15, 16, 5, 7, 12].

In this paper, our focus is on algorithms that can be directly implemented in today's commercial database systems without the addition of new access methods (which would require addressing the associated challenges of maintenance with updates, concurrency control, etc.). Specifically, we consider the Block-nested-loop Algorithm [15], and the Block-nested-loop with Presorting [5]. There are other algorithms that include `Bitmap` and `Index`[16], `NN` [7] and `BBS` [12] which use specialized data structures like extensions of B-trees and R^*-trees. We leave the cost analysis of these algorithms and indexes for future work.

2.3.1 Block-nested-loop Algorithm [15] (`BNL`)

This algorithm keeps a window of incomparable tuples in main memory. When a tuple p is read from the input, p is compared to all tuples of the window and, based on this

comparison, p is either eliminated or placed into the window. In the latter case, all the tuples in the window that p dominates are discarded. At the end, everything that remains in the window constitutes the Skyline. It is possible that the window grows bigger than the main-memory, in which case, parts of the window are written to a temporary file. Further iterations of the algorithm are required to process the temporary file.

2.3.2 Block-nested-loop with Presorting [5](SRT)

This algorithm first sorts the data using an appropriately chosen monotone scoring function and then applies the BNL algorithm. The advantage of sorting is that dominating items are likely to appear at the top and hence, the window size is expected to be small. Secondly, a data item cannot be dominated by anything thats below it. Hence, anything that is added to the window is in Skyline and can be immediately outputted. Also, only buckets need to be stored in the window rather than all data items.

3 Why Implement Skyline in the Relational Engine?

Prior work [6] has shown that while skyline does not add to the expressive power of SQL, an implementation of skyline using SQL is expensive, and that a physical implementation that is cognizant of the properties of skyline improves performance significantly. However, we are still left with the possibility that skyline is the top-most operation in an operator tree, executed after all other operations below.

This section addresses this issue. We argue from three points of view — the interaction of skyline with other operators, the mapping of the logical operator to physical implementations, and the usage of skyline in subqueries — that a tighter integration of the skyline operator into the relational server is required. We then move on to outline the challenges in implementing the skyline operator in Microsoft SQL Server 2005.

3.1 Interaction of Skyline with Other Operators

We think of the skyline operator as an aggregating operator in the spirit of the relational groupby operator. Just like groupby, the result size of skyline can vary anywhere from very few tuples to the entire input relation. Owing to this variation, pushing the skyline computation down can significantly improve performance. We illustrate this through an example.

Example 3.1 Consider a database with two relations, Cars<*make, price, year, dealer*> and Dealer<*id,location*>, where Cars.*dealer* is a foreign key pointing to Dealer.*id*. Consider the query:

```
SELECT *
FROM Cars, Dealer
WHERE make = 'Toyota' and
      Cars.dealer = Dealer.id
PREFERRING price LOW
```

The query performs a join between the Cars and Dealer tables. If we execute Skyline as the top-most operation after the join, then the full join has to be evaluated before the Skyline computation. On the other hand, if we execute the Skyline *before* the join, which is a correct transformation in this example, far fewer tuples will have to be joined with the Dealer relation. Indeed, if the prices are all unique, then exactly one tuple will pass the skyline operation, and hence if skyline is pushed down, only one tuple will have to be joined with the Dealer relation, resulting in potentially large savings.

In general, the interaction of skyline with other operators is reminiscent of the interaction of groupby with other operators. For example, a skyline operator can be pushed below a join [15] in many cases, in the same way groupby can. We review this transformation below.

Consider the relational algebra expression, $\mathcal{S}_P(T_1 \bowtie T_2)$, and suppose the following conditions hold: (i) P only refers to the attributes of T_1 and (ii) the join from T_1 to T_2 is a full join, i.e. every tuple in T_1 joins with some tuple in T_2 (for instance, a foreign key join). Then, we have the following equivalence rule:

$$\mathcal{S}_P(T_1 \bowtie T_2) \equiv (\mathcal{S}_P(T_1)) \bowtie T_2$$

We can similarly prove other equivalences that capture the interaction of skyline with other operators, similar to the equivalences involving groupby.

The key point we make from these transformations is that a cost-based analysis is needed for the query optimizer to decide whether to apply the transformation.

3.2 Mapping the Logical Operator to Physical Implementation

We next examine the issue of deciding a physical implementation for the logical skyline operator. We study this problem by comparing the running times of the algorithms introduced in Section 2.3 — the block nested-loop join algorithm (BNL) and the presorting algorithm (SRT). The SRT algorithm has an extra overhead of sorting, but after sorting it works faster than BNL. To compare their running times, we perform a simple experiment. We generate a table consisting of a million tuples, where each attribute is given a integer value randomly and independently. We vary the number of attributes and compare the running times of the two algorithms. The result is shown in Fig 1. We observe that there is no single winner and depending on the number

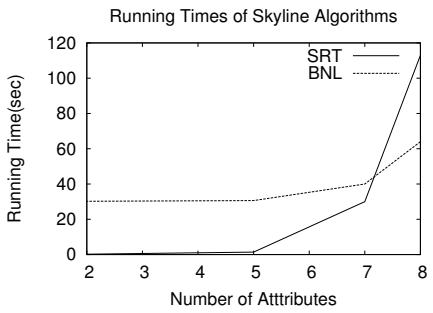

Figure 1. Running times for SRT and BNL on Numerical Data

of attributes, one algorithm significantly outperforms other. Thus, even the choice of an optimal physical implementation necessarily involves a cost based analysis.

3.3 Using Skyline in Subqueries

Finally, having skyline as an operator in the relational engine also gives the user the ability to express queries where skyline is not the top most operator, such as in a subquery. The following example query illustrates this.

```
SELECT *
FROM Cars
WHERE Cars.dealer in
   (SELECT id
    FROM Dealers
    PREFERRING (Dealers.distance LOW),
               (Dealers.rating HIGH))
```

From the above discussion, we can see that a tight integration of the skyline operator in the relational server is called for. We next address the challenges involved in such an implementation.

3.4 Challenges in Implementing Skyline in a Relational Server

Incorporating the skyline operator in a relational server involves changing the following components of a relational server. We need to add the physical implementation of the skyline operator as part of the execution engine. In addition, we need to change the query optimizer in the following ways. We first need to add the algebraic equivalences that describe the interaction of skyline with other operators. Since our implementation is in Microsoft SQL Server 2005 (Beta version), we exploit the extensible query optimizer based on the Cascades framework [4] to integrate the algebraic equivalences as transformation rules. We also make changes to the parser to expose the *Preference SQL* syntax.

But the crucial component that yields the biggest challenge is the cardinality and cost estimation modules for the skyline operator. This paper takes the first step in addressing these challenges.

Estimating the cardinality of Skyline operator is more challenging than other operators for various reasons. First, it is very sensitive to correlations among attributes. When the attributes have perfect order correlation, i.e., sorting by one implies that they are sorted by the others, the Skyline is just a single tuple. On the other hand, when attributes have perfect anti-correlation, the Skyline is the whole table. In general, it could be anywhere in between. The effect of correlation is much more significant for Skyline than for selection predicates.

Correlations in numerical attributes do occur frequently in practice. The higher the age of a car, the higher its mileage and lower its price. Thus, a Skyline query (age LOW, mileage LOW) is likely to have significantly fewer answers than what is obtained using the independence assumption. Similarly, the query (mileage LOW, price LOW) is likely to have many more answers. Also, users tend to optimize conflicting goals when specifying preferences. So user preferences are expected to have anti-correlations.

Also, for Skyline operator, a tuple belonging to the output of Skyline is dependent on every other tuple. This is unlike selection operator where one can examine a tuple in isolation and decide if it belongs to the output. A consequence of this is that standard sampling techniques, used for size estimation in presence of correlations, do not apply to Skyline.

Similarly, estimating the Skyline computation cost is more challenging than other physical operators because it is CPU intensive [15]. It is known that if there is sufficient memory to hold the Skyline, then Skyline can be computed using a single scan of the data. However, processing each tuple in memory requires a large amount of time and this dominates the cost of a single scan. For instance, on a randomly generated table with 100000 tuples, the total time taken by `BNL` algorithm on 5 and 7 dimensions was 1.2 seconds and 21 seconds respectively, while the time taken to scan the input was just 0.09 seconds.

4 Skyline Cardinality Estimation

In this section, we consider the following problem: given a table T, and a set of preferences P, compute the expected size of $\mathcal{S}_P(T)$. With abuse of notation, we will use $\mathcal{S}_P(n)$ to denote the expected size of $\mathcal{S}_P(T)$ when the size of the T is n, We will differentiate between (i) *numeric* preferences over continuous attributes, e.g. (`price` LOW)

and (`mileage` LOW) and (ii) *predicate* preferences like (`color`='red') and (`price` between 5K and 10K).

We first describe the current work on cardinality estimation, which assumes that all preferences are independent and numeric. We then show how to extend them to handle predicate preferences and correlations among attributes. In this process, we also give a generic expression that computes Skyline cardinality for any given data distribution.

4.1 Independent Numeric Preferences

The problem of estimating the size of Skyline has been considered [1, 3] when all the preferences are numeric preferences under the further assumption that all attributes are independent. The assumption of independence of attributes is standard in the estimates of most of the relational engines. Under these assumptions, $|\mathcal{S}_P(T)|$ only depends on the size of T and the number of attributes in P. Let $s(n, d)$ denote the expected size of the Skyline with d attributes and n data points. It is shown [1] that $s(n, d)$ satisfies the following recurrence:

$$s(n,d) = \frac{1}{n}s(n,d-1) + s(n-1,d) \tag{1}$$

To see why the recurrence holds, consider the tuple that has the smallest value with respect to the first preference. For this tuple to be in the Skyline, it must be in the Skyline of the remaining preferences. The probability of this is $\frac{1}{n}s(n, d-1)$ since there are n total tuples and $s(n, d-1)$ of them are in the Skyline of the remaining preferences. Also, this tuple cannot dominate any other tuple. So, out of the remaining $n-1$ tuples, $s(n-1, d)$ are expected to be in the Skyline. Hence, we get the above recurrence equation. While there is no closed form for the above formula, its known [3] that $s(n, d)$ is $\Theta((\ln n)^{d-1}/(d-1)!)$.

The above formula does not hold when the set of preferences P also contains predicate preferences. Eq (1) holds only under the assumption that no two tuples are equally good with respect to any preference, i.e. each preference totally orders the set of tuples. This is completely violated by predicate preferences which only divide the tuples into two sets: those tuples who satisfy the predicate and those who do not. Eq (1) also fails to hold when there are correlations among attributes.

4.2 A general formula

We give the formula for the cardinality of Skyline given any underlying data distribution for the tuples. We assume that the tuples are independently and identically distributed according to some distribution, though attributes may have correlations between them. Let $X_1, X_2, \cdots X_d$ be the set of attributes on which skyline is computed. Without loss of generality, we assume that the attributes take values between 0 and 1 and further, each of the attribute is minimized in the skyline. Let $F(x_1, x_2, \cdots x_d)$ denote the joint distribution function of the d variables, i.e. it denotes the probability $[X_1 \leq x_1, \cdots X_d \leq x_d]$. Similarly, let $f(x_1, \cdots x_d)$ denote the joint density function. In vector notation, we write them as $f(\bar{x})$ and $F(\bar{x})$, where $\bar{x} = (x_1, \cdots, x_d)$.

To calculate the size of Skyline, let us look at a data item t_i. Suppose it has values $\bar{x} = x_1, \cdots x_d$. Then, for t_i to be in the skyline, none of the other $n-1$ data points should have all the attributes smaller than t_i. The probability of this is $(1 - F(\bar{x}))^{n-1}$. Since t_i itself comes from a distribution with density $f(\bar{x})$, the probability that a randomly chosen data point belongs to the Skyline is

$$\int_{[0,1]^d} f(\bar{x})(1 - F(\bar{x}))^{n-1} d\bar{x}$$

Since we draw n tuples, by the linearity of expectations, we get the following result.

Theorem 4.1. *Let $t_1, t_2 \cdots t_n$ be n tuples drawn from the above probability distribution. Then, the expected value of $\mathcal{S}_P(n)$ is*

$$n \int_{[0,1]^d} f(\bar{x})(1 - F(\bar{x}))^{n-1} d\bar{x} \tag{2}$$

The above theorem is a generic result that applies to any data distribution. It gives us a tool to derive the expression for Skyline cardinality under various settings, when the preferences are numeric or predicate and independent or correlated, as we describe in the following sections.

4.3 Independent Numeric and Predicate Preferences

We now relax the condition of numeric preferences. We give an expression for the cardinality of the Skyline, still under the independence assumption, when both numeric and predicate preferences are present.

Let us start with the case when all the preferences are numeric.

Theorem 4.2. *Let $s(n, d)$ denote the cardinality of skyline over n tuples and d numeric independent preferences. Then,*

$$s(n,d) = n \int_{[0,1]^d} (1 - x_1 x_2 \cdots x_d)^{n-1} dx_1 \cdots dx_d$$

Proof. (Sketch) We apply Theorem 4.1. Observe that under independence, $f(x_1, \cdots, x_d)$ can be written as $f_1(x_1) f_2(x_2) \cdots f_d(x_d)$ and $F(x_1, \cdots, x_d)$ can be written as $F_1(x_1) \cdots F_d(x_d)$, where $f_i(x_i)$ is the density and

$F_i(x_i)$ is the distribution of attribute X_i. Further, we know that if f_i is continuous, then $f_i(x_i) = F_i'(x_i)$. Substituting this into Eq (2), and using a change of variable, we obtain the above expression for $s(n, d)$. □

Theorem 4.2 gives us an alternative expression for $s(n, d)$ that is equivalent to the solution of the recurrence in Eq (1). The integral has no closed form, but several numerical methods exist to evaluate the integral [14, 17].

Now suppose we want a Skyline when $P = \{Y_1, \cdots Y_k, X_1, \cdots X_d\}$, where Y_i are predicate preferences and X_i are numeric preferences. We assume that Y_i is 1 when the predicate is satisfied and 0 otherwise. Let p_i denote the selectivity of the predicate Y_i, i.e. the probability that $Y_i = 1$.

We need some notation. Let $v \in \{0, 1\}^k$ be any vector and let v_i denote its i component. Define

$$P_1(v) = \prod_i p_i^{v_i}(1 - p_i)^{1-v_i}$$
$$P_2(v) = \prod_i p_i^{1-v_i}$$

$P_1(v)$ denotes the probability that the attributes $\{Y_1, \cdots, Y_k\}$ have value given by the vector v. Similarly, $P_2(v)$ denotes the probability that the attributes $\{Y_1, \cdots, Y_k\}$ have value less than or equal to the vector v. We have the following result.

Theorem 4.3. *Let the set of preferences P be as defined above. Then, the expected value of $|\mathcal{S}_P(n)|$ is*

$$n \sum_{v \in \{0,1\}^d} \int_{[0,1]^d} P_1(v)(1 - P_2(v)x_1 \cdots x_d)^n dx_1 \cdots dx_d$$

The proof borrows ideas from Theorem 4.1 and 4.2. We omit the details of the proof, but instead give an example.

Example 4.4 Consider a Skyline query (`make` = 'Toyota', `year` > 2001, `price` LOW, `mileage` LOW). It has two predicate preferences and two numeric preferences. Suppose the predicate `make` = *'Toyota'* has 0.15 selectivity and the predicate `year` > *2001* has 0.4 selectivity. If all the attributes are assumed to be independent, the expected cardinality given by Theorem 4.3 is

$$n \int_{[0,1]^2} (f_1 + f_2 + f_3 + f_4) dx_1 dx_2$$

where

$$f_1(x_1, x_2) = 0.15 * 0.4(1 - x_1x_2)^n$$
$$f_2(x_1, x_2) = 0.15 * 0.6(1 - 0.4 * x_1x_2)^n$$
$$f_3(x_1, x_2) = 0.85 * 0.4(1 - 0.15 * x_1x_2)^n$$
$$f_4(x_1, x_2) = 0.85 * 0.6(1 - 0.15 * 0.4x_1x_2)^n$$

4.4 Relaxing Independence

As we have discussed, there can be large errors in the estimates of skyline cardinality in presence of correlations between attributes. The skyline size can vary from a single tuple (when attributes are perfectly correlated) to the whole relation (when attributes are perfectly anti-correlated).

We present two techniques to estimate skyline size in the presence of correlations, *sampling* and *histograms*.

Sampling Sampling techniques for selectivity estimation [11] and query size estimation in general [10] have been known for a long time.

A naive method of sampling is as follows: compute the Skyline size on a small random sample of the data and scale it linearly to the actual data size. This does not work for the reason that the size of the Skyline is not a linear function of the data size. However, sampling still tells us something: if the sample has larger Skyline than what we would expect assuming independence, there must be anti-correlations among attributes and the Skyline on the whole data must also be larger that expected.

It is known than when the attributes are independent, the skyline size on d attributes is $\Theta(\log^d n)$. We found that in presence of correlations/anti-correlations that are not very large, the skyline size still grows as some power of $\log n$ (see Section 6). We use this hypothesis to estimate the skyline cardinality in the following way: assume that the skyline size on a give set of attributes is $A \log^B n$. Compute the skyline on a small sample of the data to estimate the parameters A and B and use them to calculate the size of skyline on the whole data. Note that this does not hold near perfect anti-correlations, when the skyline contains all tuples. However, it still gives us a much better estimate than simply using independence assumption. We leave the theoretical investigation of this behavior for future work.

Histograms Histograms have been extensively studied in the database community for query size estimation. There have been various techniques [9, 13] for using and efficiently constructing multidimensional histograms to model attribute correlations.

In the context of Skyline size estimation, we can use these structures to estimate the joint distribution function of the attributes, i.e. the functions $F(\bar{x})$ and $f(\bar{x})$ in Eq 2. The idea is to divide the joint domain of all the attributes into small buckets and maintain a histogram that counts the number of points in each bucket. We can then approximate $f(\bar{x})$ on a bucket as the fraction of the points in that bucket and $F(\bar{x})$ as the fraction of points that lie in buckets below the given bucket. Thus, we can approximate the integral in $Eq\ (2)$. An alternative to maintaining joint histograms is to use sampling to estimate the number of points in each bucket, and use that in turn to approximate the joint distribution.

5 Skyline Cost Estimation

Costing the Skyline operator is more challenging than other physical operators because it is CPU intensive. If there is sufficient memory to hold the Skyline, then Skyline can be computed using a single scan of the data. However, processing each tuple in memory requires a large amount of time as each tuple is compared with all the maximal tuples found so far.

5.1 Cost estimation for BNL algorithm

We consider the basic BNL algorithm and assume that the Skyline operator completes after a single scan of data. There are two components to the cost: the I/O cost of scanning the data and the CPU cost of processing it. The I/O cost can be computed in a straightforward manner from the size of the relation. The CPU cost is directly proportional to the number of comparisons performed in memory. In this section, we will derive an expression for the expected number of comparisons performed by BNL algorithm.

Consider the operator $\mathcal{S}_P(T)$ that computes Skyline over table T with respect to the set of preferences P. P may contain both numeric and predicate preferences. We assume that there is some underlying data distribution for the tuples in T, which may have correlations between attribute values. We assume that the tuples in table T are not laid out in any particular order, so that the tuples that constitute the Skyline have equal chance of occurring anywhere in the table. Let $s(n, P)$ denote the function that gives the expected value of $|\mathcal{S}_P(T)|$ when T contains n tuples selected from the underlying data distribution. Similarly, let $c_{\texttt{BNL}}(n, P)$ denote the expected number of comparisons performed by BNL. We will derive an expression for $c_{\texttt{BNL}}(n, P)$ in terms of $s(n, P)$.

Let the data items be numbered from 1 to n. For $1 \leq i < j \leq n$, let $P_{i,j}$ denote the probability that the data items i and j are compared. For this to happen, i should be in the memory when j comes and j should not get discarded before it is compared with i. For the former to happen, i must be in the Skyline of the first $j-1$ data items. The probability of this is $\frac{s(j-1,P)}{j-1}$ since $s(j-1, P)$ out of the $j-1$ elements are expected in the Skyline, and we assume that tuples in T are ordered randomly. For the latter to happen, j should be in the Skyline of data items $i+1$ to $j-1$. This is because j will be compared to all the data items numbered greater than i before it is compared to i. The probability of this, using the same argument, is $\frac{s(j-i-1,P)}{j-i-1}$. We approximate $P_{i,j}$ using the following approximation:

$$P_{i,j} \approx \frac{s(j-1,P)}{j-1} * \frac{s(j-i-1,P)}{j-i-1}$$

Note that the equality need not be exact because the event that i is in the skyline of first $j-1$ data items is not independent from the event that j is in the skyline of $i+1$ to $j-1$. However, we empirically show that this approximation works well in practice.

Finally, by linearity of expectations, the expected number of comparisons is

$$c_{\texttt{BNL}}(n, P) \quad = \quad \sum_{i=1}^{n} \sum_{j=i+1}^{n} P_{i,j}$$

Substituting the expression for $P_{i,j}$ in the equation, we get the following expression for the expected number of comparisons performed by BNL algorithm on n data points is

$$c_{\texttt{BNL}}(n, P) \approx \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{s(j-1,P)}{j-1} * \frac{s(j-i-1,P)}{j-i-1} \quad (3)$$

where $s(x, P)$ denotes the expected cardinality of Skyline for the preferences P on x data points.

Note that we did not use independence of attributes in the above derivation. The expression is applicable irrespective of whether we have numeric or predicate preferences and whether the attributes are correlated. We just need the correct expression for $s(n, P)$.

Approximating $c_{\texttt{BNL}}(n, P)$: Eq (3) gives the exact expression for the expected number of comparisons performed by BNL algorithm. When n is large, the computation becomes expensive. However, the following method can be used to efficiently approximate the cost. If we denote $f(i, j)$ the function $\frac{s(j-1,P)}{j-1} * \frac{s(j-i-1,P)}{j-i-1}$, then $c_{\texttt{BNL}}(n, P)$ is $\sum_{i=1}^{n} \sum_{j=i+1}^{n} f(i, j)$. To approximate this function, we divide the domain of the summation into small parts, and in each part, we assume $f(i, j)$ to be a constant. This is analogous to the mid-point method for numerical integration, and gives the following expression:

$$c_{\texttt{BNL}}(n, P) \approx \sum_{i=1}^{k} \sum_{j=i+1}^{k} k^2 * f(i\frac{n}{k} + \frac{k}{2}, j\frac{n}{k} + \frac{k}{2})$$

As a final remark, if a Skyline contains only numeric preferences, we can use Eq 1 to further simplify Eq (3) to get the following:

$$c_{\texttt{BNL}}(n, P) \approx \sum_{j=2}^{n} \frac{s(j-1,d)}{j-1} s(j-1, d+1)$$

5.2 Cost Estimation of SRT algorithm

The SRT algorithm consists of a sorting phase followed by skyline computation. The cost of sorting can be computed using standard existing techniques. The skyline

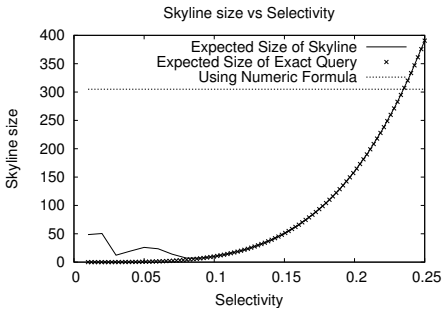

Figure 2. Expected Skyline size for predicate preferences

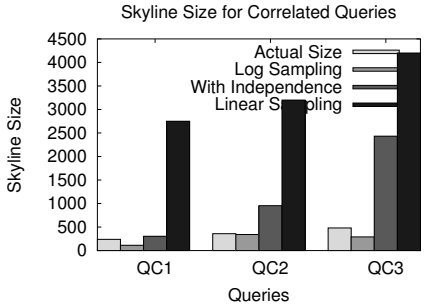

Figure 3. Expected Skyline size for correlated queries

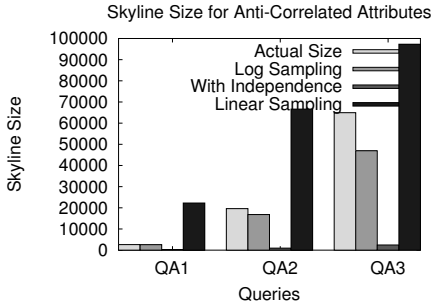

Figure 4. Expected Skyline size for anti-correlated queries

cost, similar to the BNL algorithm, requires an estimate of the number of comparisons performed in memory. Let $c_{\mathtt{SRT},A}(n, P)$ denote the expected number of comparisons performed by a Skyline on n points with P as the set of preferences, when the data is sorted on attribute A.

Let $s(x, P)$ denote the expected cardinality of Skyline for the preferences P on x data points. Suppose the data is sorted on attribute A. Let P' be the preferences obtained by deleting from P the preference on A. If A is not order-correlated with the rest of the attributes, we have

$$c_{\mathtt{SRT}}(n, P) \approx \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{s(j-1, P')}{i-1} * \frac{s(j-i-1, P')}{j-i-1}$$

Thus, sorting the data on an attribute roughly incurs the cost of computing the Skyline on remaining attributes.

6 Experiments

Effect of Predicate Preferences In the first set of experiments, we study the effect of predicate preferences over the cardinality of Skyline. We consider a table T with 5 attributes and with $n = 100000$ tuples. We consider a Skyline query consisting of 5 independent predicate preferences, each having a selectivity p. Figure 2 shows the plots of expected size of the Skyline as a function of p.

Note that the Skyline size is a direct function of the selectivity. On the other hand, if we ignore the fact that the Skyline has predicate preferences, and use the expression for numerical preferences, we get a constant number, which is plotted in the above figure.

It is also interesting to compare the Skyline size with the expected size of the exact query, i.e. the query that wants all the predicates to be satisfied. Since we have 5 independent predicates each with selectivity p, the expected size of the query is $n * p^5$. Now, if there is at least one data item that satisfies all the predicates, all such data items define the Skyline. So, if $n * p^5 \geq 1$, we should expect the Skyline size to be very close [1] to $n * p^5$. Figure 2 shows that indeed the two curves coincide for large values of p. The figure also shows that for smaller values of p, the Skyline size is larger that the expected query size, but not too large. We conclude that for practical purposes, if the Skyline query contains only predicate preferences, we can use the exact query size (which is much easier to estimate by the database) to approximate the Skyline size.

Sampling for Cardinality Estimation In order to have a controlled environment we created a simple benchmark of data and queries. We constructed a single table $T(a_1, \cdots, a_9)$ having 9 numeric attributes. Attributes a_1, a_2 and a_3 were generated independently, a_4, a_5 and a_6 were correlated to the first three attributes and a_7, a_8 and a_9 were anti-correlated to the first three attributes. We considered six queries, each specifying numeric preferences over a sub-

[1] The expected Skyline size will not be exactly $n * p^5$ because even if $n * p^5 \geq 1$, there is a finite but very small probability that no item has all attributes 1. This probability will quickly converge to 0 as p increases

set of attributes as described below:

$$
\begin{aligned}
QC_1 &: a_1, a_2, a_3, a_4 \\
QC_2 &: a_1, a_2, a_3, a_4, a_5 \\
QC_3 &: a_1, a_2, a_3, a_4, a_5, a_6 \\
QA_1 &: a_1, a_2, a_3, a_7 \\
QA_2 &: a_1, a_2, a_3, a_7, a_8 \\
QA_3 &: a_1, a_2, a_3, a_7, a_8, a_9
\end{aligned}
$$

We used three methods to estimate the size of Skyline in each case:

1. Using Independence: we assume that all attributes are independent and use Eq 1 to estimate the size.
2. Using Naive Sampling: we compute the Skyline on a small random sample of the table, and scale it linearly to the table size.
3. Using Log Sampling, as described in Section 4.4. The size of the random sample is 1% of the total table size.

Figure 3 shows the results of the three techniques on the correlated queries QC_1, QC_2 and QC_3. Figure 4 shows the same graph for the anti-correlated queries QA_1, QA_2 and QA_3.

For correlated queries, the Skyline size is much smaller than that using independence assumption, while for anticorrelated queries, the size is much larger. In both the cases, log-sampling gives a reasonable estimate of the correct size. Note that a simple linear scaling always overestimates by a large factor. This is because the Skyline size is a sub-linear function of the table size in all these cases.

Cost Estimates for `BNL` and `SRT` algorithms: We consider a Skyline on a table with 7 independent numeric attributes, and we plot the expected number of comparisons as a function of the number of tuples using our cost estimates. Figure 5 shows this graph for `BNL` algorithm and Figure 6 shows the same graph for `SRT`. The graph validates our cost estimates. It also shows that the variation in Skyline size is small, so the expected Skyline size is within tolerable error bounds of the actual size.

Impact on Actual Plans Finally, we study the impact of our cardinality and cost estimation techniques on the plans produced by the query optimizer, and the resulting running times of skyline queries in a real implementation. For this purpose, we implement the skyline operator in Microsoft SQL Server 2005 (Beta version). We implement both the physical implementations addressed in this paper, and all the transformation rules discussed in Section 3. We implement the cardinality estimation based on attribute value independence, which does not need the maintenance of any synopsis structures. For the sampling based approach (both naive and log-scaling), we maintain samples of base

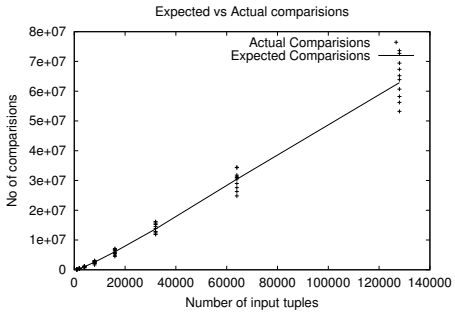

Figure 5. Costing of BNL algorithm

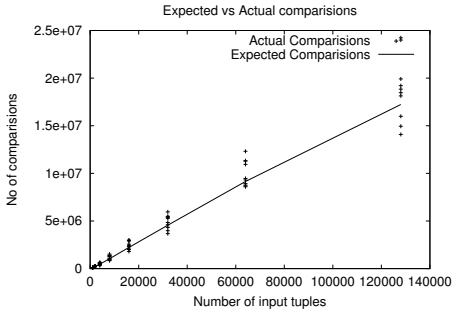

Figure 6. Costing of SRT algorithm

tables and use the sampling based cardinality estimation only when skyline is the leaf operator. The problem of propagating the cardinality estimates given base table samples is left for future work.

We use the data described earlier in this section. Since we want to study the impact on actual plans, we modify the queries to include joins. We create a large table U in the database that fully joins with T. We also create an unclustered index on the joining columns in U.

We consider a skyline on $T \bowtie U$ where the preferences are expressed over the following columns in T: $a_1, a_2, a_3, a_4, a_5, a_6$. Note that this kind of join is quite realistic. For instance, T could be a table of cars and U could be a table of dealers, and the above join would express preferences on cars and find the corresponding dealers.

We first find that for all cardinality estimation methods, the skyline gets pushed below the join to be evaluated on T before it joins with U. This illustrates the benefit of implementing the skyline operator in the server, where the interaction with other operators can yield to efficient plans.

The question arises which join method the optimizer picks to implement $T \bowtie U$. If the skyline cardinality is small, then it goes with an index nested loops implementation. On the other hand, if the skyline cardinality is large, it picks a hash join since an index nested loops would result

in too many unclustered index lookups.

Since the columns a_4, a_5, a_6 are correlated to the first three attributes, the real skyline cardinality is small — the skyline has 481 tuples. For this value of the cardinality, the optimal plan is based on index nested loops.

However, assuming attribute value independence tends to overestimate the skyline cardinality to be 2432 (note that this is in contrast with traditional selectivity estimation where attribute value independence typically tends to underestimate selectivities). The naive sampling approach also overestimates the skyline cardinality. Hence, when we use either of these cardinality estimation methods, the query optimizer picks a hash join to implement $T \bowtie U$.

On the other hand, log sampling estimates the cardinality to be 290 and the query optimizer picks an index nested loops join. In terms of running times, the hash join turns out to be slower by a factor of 1.62 times.

This experiment illustrates that using log sampling to capture correlations yields substantial benefits in actual running times for skyline queries.

7 Related Work

The problem of computing the Skyline has been studied under the name of maximum vector problem [8]. Earlier work [8, 1] on computing the Skyline was algorithmic in nature where all the data was assumed to be available in memory. Several algorithms suitable for a database system have recently been proposed [15, 16, 5, 7]. Some of the algorithms use special index structures likes variants of B-trees [16] or R-trees [7, 12]. Chomicki et al. observed that a simple sorting of data before computing Skyline can substantially reduce the cost of Skyline computation. The problem of estimating the size of Skyline has been considered [1, 3] under the assumption that no two data items have the same value on any attribute and all attributes are independent. Chan et al. [2] show how to evaluate skylines efficiently over partially-ordered domains. Kossman [15] gives some optimizations for Skyline over joins. Kiebling et al. [6] propose an extension of SQL language to include user preferences.

8 Conclusions

In this paper, we considered the issues with the implementation of Skyline as a operator in relation engine. We showed that the current methods for cardinality estimations are not adequate for a real database implementation and we provided robust techniques for this purpose. We considered two physical algorithms discussed in the literature, the Block-nested-loop algorithms and the Presorting algorithms, and we showed how these algorithms can be costed. Finally, we implemented our techniques in a prototype Microsoft SQL Server and showed that Skyline queries can significantly benefit from such an implementation.

References

[1] J. L. Bentley, H. T. Kung, M. Schkolnick, and C. D. Thompson. On the average number of maxima in a set of vectors and applications. *J. ACM*, 25(4):536–543, 1978.

[2] C.-Y. Chan, P.-K. Eng, and K.-L. Tan. Stratified computation of skylines with partially-ordered domains. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 203–214, 2005.

[3] P. Godfrey. Skyline cardinality for relational processing. In *Foundations of Information and Knowledge Systems*, 2004.

[4] G. Graefe. The cascades framework for query optimization. *IEEE Data Eng. Bull.*, 18(3):19–29, 1995.

[5] J. G. Jan Chomicki, Parke Godfrey and D. Liang. Skyline with presorting. In *Proceedings of the 19th Int. Conf. on Data Engineering, Banglore, India*, 2003.

[6] W. Kiebling and G. Kostler. Preference sql - design, implementation, experiences. In *VLDB*, pages 990–1001, 2002.

[7] D. Kossmann, F. Ramsak, and S. Rost. Shooting stars in the sky: An online algorithm for skyline queries. In *VLDB*, 2002.

[8] H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *J. ACM*, 22(4):469–476, 1975.

[9] J.-H. Lee, D.-H. Kim, and C.-W. Chung. Multi-dimensional selectivity estimation using compressed histogram information. In *SIGMOD*, pages 205–214, 1999.

[10] R. J. Lipton and J. F. Naughton. Query size estimation by adaptive sampling. In *PODS*, pages 40–46, 1990.

[11] R. J. Lipton, J. F. Naughton, and D. A. Schneider. Practical selectivity estimation through adaptive sampling. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data, Atlantic City, NJ, May 23-25, 1990*, pages 1–11, 1990.

[12] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An optimal and progressive algorithm for skyline queries. In *SIGMOD '03: Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pages 467–478, 2003.

[13] V. Poosala and Y. E. Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB '97: Proceedings of the 23rd International Conference on Very Large Data Bases*, pages 486–495, 1997.

[14] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling. *Numerical Recipes in FORTRAN: The Art of Scientific Computing*. Cambridge University Press, 3rd edition, 1992.

[15] D. K. Stephan Brzsnyi and K. Stocker. The skyline operator. In *Proceedings of the 17th International Conference on Data Engineering, Heidelberg, Germany*, 2001.

[16] K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 301–310, 2001.

[17] C. W. Ueberhuber. *Numerical Computation 2: Methods, Software, and Analysis*. Springer-Verlag, 1997.