Skyline with Presorting: Theory and Optimizations

Jan Chomicki[1], Parke Godfrey[2], Jarek Gryz[2], and Dongming Liang[2]

[1] University at Buffalo, USA
[2] York University, Canada

Abstract. There has been interest recently in skyline queries, also called Pareto queries, on relational databases. Relational query languages do not support search for "best" tuples, beyond the `order by` statement. The proposed skyline operator allows one to query for best tuples with respect to any number of attributes as preferences. In this work, we explore what the skyline means, and why skyline queries are useful, particularly for expressing preference. We describe the theoretical aspects and possible optimizations of an efficiant algorithm for computing skyline queries presented in [6].

1 Introduction and Motivation

Often one would like to query a relational database in search of a "best" match, or tuples that best match one's preferences. Relational query languages provide only limited support for this: the `min` and `max` aggregation operators, which act over a single column; and the ability to *order* tuples with respect to their attribute values. In **SQL**, this is done with the `order by` clause. This is sufficient when one's preference is synonymous with the values of one of the attributes, but is far from sufficient when one's preferences are more complex, involving more of the attributes.

Consider a table of restaurant guide information, as in Figure 1. Column **S** stands for *service*, **F** for *food*, and **D** for *decor*. Each is scored from 1 to 30, with 30 as the best. We are interested in choosing a restaurant from the guide, and we are looking for a best choice, or best choices from which to choose. Ideally, we would like the choice to be the best for service, food, *and* decor, *and* be the lowest priced. However, there is no restaurant that is better than all others on every criterion individually, as is usually the case in real life, and in real data. No one restaurant "trumps" all others. For instance, Summer Moon is best on food, but Zakopane is best on service.

While there is no one best restaurant with respect to our criteria, we want at least to eliminate from consideration those restaurants which are worse on all criteria than some other. Thus, the Briar Patch BBQ should be eliminated because the Fenton & Pickle is better on all our criteria and is thus a better choice. The Brearton Grill is in turn eliminated because Zakopane is better than it on all criteria. Meanwhile the Fenton & Pickle is worse on every criterion than every other (remaining) restaurant, except on price, where it

restaurant	S	F	D	price
Summer Moon	21	25	19	47.50
Zakopane	24	20	21	56.00
Brearton Grill	15	18	20	62.00
Yamanote	22	22	17	51.50
Fenton & Pickle	16	14	10	17.50
Briar Patch BBQ	14	13	3	22.50

Fig. 1. Example restaurant guide table, GoodEats.

is the best. So it stays in consideration. This would result in the choices in Figure 2.

restaurant	S	F	D	price
Summer Moon	21	25	19	47.50
Zakopane	24	20	21	56.00
Yamanote	22	22	17	51.50
Fenton & Pickle	16	14	10	17.50

Fig. 2. Restaurants in the skyline.

In [3], a new relational operator is proposed which they name the *skyline operator*. They propose an extension to **SQL** with a `skyline of` clause as counterpart to this operator that would allow the easy expression of the restaurant query we imagined above. In [9] and elsewhere, this is called the *Pareto operator*. Indeed, the notion of Pareto optimality with respect to multiple parameters is equivalent to that of choosing the non-dominated tuples, designated as the skyline.

```
select ... from ... where ...
    group by ... having ...
    skyline of a[1][min|max|diff], ..., a[n]
```

Fig. 3. A proposed skyline operator for SQL.

The `skyline of` clause is shown in Figure 3. Syntactically, it is similar to an `order by` clause. The columns $a_1, \ldots, a_n$ are the attributes that our preferences range over. They must be of domains that have a natural total ordering, as integers, floats, and dates. The directives `min` and `max` specify whether we prefer low or high values, respectively. The directive `diff` says that we are interested in retaining best choices with respect to every distinct value of that attribute. Let `max` be the default directive if none is stated. The

skyline query in Figure 4 over the table `GoodEats` in Figure 1 expresses what we had in mind above for choosing "best" restaurants, and would result in the answer set in Figure 2.

```
select * from GoodEats
    skyline of S max, F max, D max, price min
```

Fig. 4. Skyline query to choose restaurants.

Skyline queries are not outside the expressive power of current **SQL**. The query in Figure 5 shows how we can write an arbitrary skyline query in present **SQL**. The c_i's are attributes of `OurTable` that we are interested to retain in our query, but are not skyline criteria. The s_i are the attributes that are our skyline criteria to be maximized, and would appear in `skyline of` as s_i `max`. (Without loss of generality, let us only consider `max` and not `min`.) The d_i are the attributes that are the skyline criteria to *differ*, and would appear in `skyline of` as d_i `diff`.

```
select c_1, ..., c_k, s_1, ..., s_m, d_1, ..., d_n
       from OurTable
except
select D.c_1, ..., D.c_k, D.s_1, ..., D.s_m, D.d_1, ..., D.d_n
    from OurTable T, OurTable D
    where D.s_1 ≤ T.s_1 and ... D.s_m ≤ T.s_m and
          (D.s_1 < T.s_1 or ... D.s_m < T.s_m)and
          D.d_1 = T.d_1 and ... D.d_n = T.d_n
```

Fig. 5. SQL for generating the skyline set.

Certainly it would be cumbersome to need to write skyline-like queries in this way. The skyline clause is a useful syntactic addition. More important than ease of expression, however, is the expense of evaluation. The query in Figure 5 can be quite expensive. It involves a self-join over a table, and this join is a θ-join, not an equality-join. The self-join effectively computes the tuples that are trumped—or *dominated*—by other tuples. The tuples that remain, that were never trumped, are then the skyline tuples. It is known that the size of the skyline tends to be small, with certain provisos, with respect to the size of the table [7]. Thus, the intermediate result-set before the `except` can be enormous.

No current query optimizer would be able to do much with the query in Figure 5 to improve performance. If we want to support skyline queries, it is necessary to develop an efficient algorithm for computing skyline. And if we want the skyline operator as part of **SQL**, this algorithm must be

easy to integrate in relational query plans, be well-behaved in a relational context, work in all cases (without special provisions in place), and be easily accommodated by the query optimizer.

Recent years have brought new interest in expressing preference queries in the context of relational databases and the World Wide Web. Two competing approaches have emerged so far. In the first approach [1,8], preferences are expressed by means of preference (utility) functions. The second approach uses logical formulas [4,9] and, in particular, the `skyline` operator [3,12], described in the previous section. Skyline computation is similar to the *maximal vector problem* studied in [2,10,11]. These consider algorithmic solutions to the problem and address the issue of skyline size. None of these works addresses the problem in a database context, however. In [7], we address the question of skyline query cardinality more concretely.

In this paper, we explore what the skyline means, and why skyline queries are useful, particularly for expressing preference. We describe a well-behaved, efficient algorithm for computing skyline queries. Our algorithm improves on exisiting approaches in efficiency, pipelinabilty of output (of the skyline tuples), stability of run-time performance, and being applicable in any context.

2 Skyline versus Ranking

The skyline of a relation in essence represents the best tuples of the relation, the Pareto optimal "solutions", with respect to the skyline criteria. Another way to find "best" tuples is to score each tuple with respect to one's preferences, and then choose those tuples with the best score (ranking). The latter could be done efficiently in a relational setting. In one table scan, one can score the tuples *and* collect the best scoring tuples.

How is skyline related to ranking then? It is known that the skyline represents the closure over the maximum scoring tuples of the relation with respect to all *monotone scoring functions*. For example, in choosing a restaurant as in the example in Section 1, say that one values service quality twice as much as food quality, and food quality twice as much as decor, those restaurants that are best with respect to this "weighting" will appear in the skyline. Furthermore, the skyline is the least-upper-bound closure over the maximums of the monotone scoring functions [3].

This means that the skyline can be used instead of ranking, or it can be used in conjunction with ranking. First, since the best tuples with respect to any (monotone) scoring are in the skyline, one only needs effectively to query the skyline with one's preference queries, and not the original table itself. The skyline is (usually) significantly smaller than the table itself [7], so this would be much more efficient if one had many preference queries to try over the same dataset. Second, as defining one's preferences in a preference query can be quite difficult, while expressing a skyline query is relatively easy, users may find skyline queries beneficial. The skyline over-answers with respect to

the users' intent in a way, since it includes the best tuples with respect to *any* preferences. So there will be some choices (tuples) among the skyline that are not of interest to the user. However, every best choice with respect to the user's implicit preferences shows up too.

While in [3], they observe this relation of skyline with monotone scoring functions, they did not offer proof nor did they discuss *linear scoring functions*, to which much work restricts focus. Let us investigate this more closely, and more formally, then, for the following reasons:

- to relate skyline to preference queries, and to illustrate that expressing preferences by scoring is more difficult than one might initially expect;
- to rectify some common misconceptions regarding scoring for the purposes of preference queries, and regarding the claim for skyline; and
- to demonstrate a useful property of monotone scoring that we can exploit for an efficient algorithm to compute the skyline.

Let attributes $a_1, \ldots, a_k$ of schema R be the skyline criteria, without loss of generality, with respect to "max". Let the domains of the a_i's be real, without loss of generality. Let $\mathbf{R}$ be a relation of schema R, and so represents a given instance.

Definition 1. Define a *monotone scoring function* S with respect to R as a function that takes as its input domain tuples of R, and maps them onto the range of reals. S is composed of k monotone increasing functions, $f_1, \ldots, f_k$. For any tuple $t \in \mathbf{R}$, $S(t) = \sum_{i=1}^{k} f_i(t[a_i])$.

Lemma 1. *Any tuple that has the best score over* $\mathbf{R}$ *with respect to any monotone scoring function* S *with respect to* R *must be in the skyline.*[1]

It is more difficult to show that every tuple of the skyline is the best score of some monotone scoring. Most restrict attention to linear weightings when considering scoring, though, so let us consider this first.

Definition 2. Define a *positive, linear scoring function*, W, as any function over a table $\mathbf{R}$'s tuples of the form $W(t) = \sum_{i=1}^{k} w_i t[a_i]$, in which the w_i's are positive, real constants.

As we insist that the w_i's are positive, the class of the positive, linear scoring functions is a proper sub-class of the monotone scoring functions. Commonly in preference query work, as in [8], the focus is restricted to linear scoring. It is not true, however, that every skyline tuple is the best with respect to some positive, linear scoring.

Theorem 1. *It is possible for a skyline tuple to exist on* $\mathbf{R}$ *such that, for every positive, linear scoring function, the tuple does not have the maximum score with respect to the function over table* $\mathbf{R}$.

[1] Proofs of all lemmas and theorems can be found in [5].

Consider $\mathbf{R} = ((4,1),(2,2),(1,4))$. All three tuples are in the skyline (`skyline` of a_1, a_2). Linear scorings that choose (4,1) and (1,4) are obvious, but there is no positive, linear scoring that scores (2,2) best. Note that (2,2) is an interesting choice. Tuples (4,1) and (1,4) represent in a way outliers. They make the skyline because each has an attribute with an extrema value. Whereas (2,2) represents a balance between the attributes (and hence, preferences). For example, if we are conducting a house hunt, a_1 may represent the number of bathrooms, and a_2, the number of bedrooms. Neither a house with four bathrooms and one bedroom, nor one with one bathroom and four bedrooms, seem very appealing, whereas a 2bth/2bdrm house might.

Theorem 2. *The skyline contains all, and only, tuples yielding maximum values of monotone scoring functions.*

While there exists a monotone scoring function that chooses—assigns the highest score to—any given skyline tuple, it does not mean anyone would ever find this function. In particular, this is because, in many cases, any such function is a contrivance based upon that skyline's values. The user is searching for "best" tuples and has not seen them yet. Thus, it is unlikely anyone would discover a tuple like (2,2) above with any preference query. Yet, the 2bth/2bdrm house might be exactly what we wanted.

For the algorithm for skyline computation we are to develop, we can exploit our observations on the monotone scoring functions. Let us define the *dominance relation*, "$\preceq$", as follows: for tuples any $r, t \in \mathbf{R}$, $r \preceq t$ *iff* $r[a_i] \leq t[a_i]$, for all $i \in 1, \ldots, k$. Further define that $r \prec t$ *iff* $r \preceq t$ *and* $r[a_i] < t[a_i]$, for some $i \in 1, \ldots, k$.

Theorem 3. *Any total order of the tuples of* $\mathbf{R}$ *with respect to* any *monotone scoring function (ordered from highest to lowest score) is a topological sort with respect to the skyline dominance partial relation ("$\preceq$").*

Consider the total ordering on $\mathbf{R}$ provided by the basic **SQL** `order by` as in the query in Figure 6. This total order is a topological sort with respect to dominance.

```
select * from R
    order by a_1 desc, ..., a_k desc;
```

Fig. 6. An order by query that produces a total monotone order.

The following proposition is fairly obvious and it is used to build a better skyline algorithm.

Theorem 4. *Any nested sort of* $\mathbf{R}$ *over the skyline attributes (sorting in descending order on each), as in the query in Figure 6, is a topological sort with respect to the dominance partial order.*

As we read the tuples output by the query in Figure 6 one by one, it is only possible that the current tuple is dominated by one of the tuples that came before it (if, in fact, it is dominated). It is impossible that the current tuple is dominated by any tuple to follow it in the stream. Thus, the very first tuple must belong to the skyline; no tuple precedes it. The second tuple might be dominated, but only by first tuple, if at all. And so forth.

The last observation provides us the basis for an algorithm to compute skyline (the details of the algorithm, called **SFS**, are presented in [6]). First, we sort our table as with the query in Figure 6. In a relational engine, an external sort routine can be called for this. Buffer pool space is then allocated as a *window* in which skyline tuples are to be placed as found. A cursor pass over the sorted tuples is then commenced. The current tuple is checked against the tuples cached in the window. If the current tuple is dominated by any of the window tuples, it is safe to discard it. It cannot be a skyline tuple. (We have established that the current tuple cannot dominate any of the tuples in the window.) Otherwise, the current tuple is incomparable with each of the window tuples. Thus, it is a skyline tuple itself. Note that it was sufficient that we compared the current tuple with just the window tuples, and not all tuples that preceded it. This is because if any preceding tuples were discarded, it can only be because another tuple already in the window dominated it. Since dominance is transitive, then comparing against the window tuples is sufficient. In the case that the current tuple was not dominated, if there is space left in the window, it is added to the window. Note that we can also place the tuple on the output stream simultaneously, as we know that it is skyline. The algorithm fetches the next tuple from the stream and repeats.

3 Optimizations

Reduction Factor

A key to efficiency for any skyline algorithm is to eliminate tuples that are not skyline as quickly as possible. In the ideal, every eliminated tuple would only be involved in a single comparison, which shows it to be dominated. In the worst case, a tuple that is eventually eliminated is compared against every other tuple with which it is incomparable (with respect to dominance) before it is finally compared against a tuple that dominates it. In cases that **SFS** is destined to make multiple passes, how large the run of the second pass will be depends on how efficient the window was during the first pass at eliminating tuples.

For **SFS** only skyline tuples are kept in the window. One might think on first glance that any skyline tuple ought to be good at eliminating other tuples, that it will likely dominate many others in the table. This is not necessarily true, however. Recall the definition of a skyline tuple: a tuple that is *not* dominated by any other. So while some skyline tuples are great dominators, there are possibly others that dominate *no* other tuples.

Let us formalize this some, for sake of discussion. Define a function over the domain of tuples in $\mathbf{R}$ with respect to $\mathbf{R}$ called the *dominance number*, *dn*. This function maps a tuple to the number of tuples in $\mathbf{R}$ that it properly dominates ("$\prec$"). So, given that $\mathbf{R}$ has n tuples, $dn(t)$ can range from 0 to $n-1$. If tuple t is in the window for the complete first pass, at least $dn(t)$ tuples will be eliminated. Of course, *dn*'s are not additive: window tuples will dominate some of the same tuples in common. However, this provides us with a good heuristic: We want to maximize the cumulative *dn* of the tuples in the window. This will tend to maximize the algorithm's reduction factor.

Once the window is filled on a pass for **SFS**, the cumulative *dn* is fixed for the rest of the pass. Our only available strategy is to fill the window initially with tuples with high *dn*'s. This is completely dependent upon the sort order of the tuples established before we commence the filtering passes. Let us analyze what happens currently. We employ a sort as with the query in Figure 6, a nested sort over the skyline attributes. The very first tuple t_1 (which must be skyline) has the maximum a_1 value with respect to $\mathbf{R}$. Say that $t_1[a_1] = 100$. Then $t_1[a_2]$ is the maximum with respect to all tuples in $\mathbf{R}$ that have $a_1 = 100$. This is probably high. And so forth for $a_3, ..., a_k$. Thus, t_1 with high probability has a high *dn*. Now consider t_i such that $t_i[a_1] = 100$, but $t_{i+1}[a_1] < 100$. So t_i is the last of the "$a_1 = 100$" group. Its a_2 value is the *lowest* of the "$a_1 = 100$" group, and so forth. With high probability, t_i's *dn* is low. However, if t_i is skyline (and it well could be), it is added to the window.

So **SFS** using a nested sort for its input tends to flood the window with skyline tuples with low *dn*'s, on average, which is the opposite of what we want. In Section 2, we observed that we can use *any* monotone scoring function for sorting as input to **SFS**. It might be tempting, if we could know tuples' *dn*'s, to sort on *dn*. The *dn* function is, of course, monotone with respect to dominance. However, it would be prohibitively expensive to calculate tuples' *dn*'s. Next best then would be to approximate the *dn*'s, which we can do.

Instead of a tuple's *dn*, we can estimate the probability that a given tuple dominates an arbitrary tuple. For this, we need a model of our data. Let us make the following assumptions. First, each skyline attribute's domain is over the reals between 0 and 1, non-inclusive. Second, the values of an attribute in $\mathbf{R}$ are uniformally distributed. Lastly, the values of the skyline attributes over the tuples of $\mathbf{R}$ are pair-wise independent. So given a tuple t and a randomly chosen $r \in \mathbf{R}$, what is the probability that $t[a_i] > r[a_i]$? It is the value $t[a_i]$ itself, due to our uniform distribution assumption (and due to that $t[a_i]$ is normalized between 0 and 1). Then the probability that $r \prec t$, given t is $\prod_{i=1}^{k} t[a_i]$ by our independence assumption. We can compute this for each tuple just from the tuple itself. Is this probability a monotone scoring function? It is easy to show that it is monotone. However, it is not formally a monotone scoring function as we defined this in Section 2; the definition only allowed

addition of the monotone functions applied over the skyline attributes. Define the monotone scoring function E then as $E(t) = \sum_{i=1}^{k} \ln(t[a_i] + 1)$. This clearly results in the same order as ordering by the probability. Interestingly, this is an *entropy* measure, so let us call this monotone scoring function E *entropy scoring*.

Our first assumption can always be met by normalizing the data. Relational systems usually keep statistics on tables, so it should be possible to do this without accessing the data. The second assumption of uniform distribution of values is often wrong. However, we are not interested in the actual dominance probability of tuples, but in a relative ordering with respect to that probability. Other distributions would not effect this relative ordering much, so E would remain a good ordering heuristic in these cases. The last assumption of independence too is likely to be wrong. Even in cases where independence is badly violated, E should remain a good heuristic, as again, the relative ordering would not be greatly effected. Regardless of the assumptions, E is always a monotone scoring function over $\mathbf{R}$, and we can always safely use it with **SFS**.

Projection

For **SFS**, a tuple is added to the window only if it is in the skyline. Therefore, the tuple at the same time can be pushed to the output. So it is not necessary to keep the actual tuple in the window. All we need is that we can check subsequent tuples for whether they are dominated by this tuple. For this, we only need the tuple's skyline attributes. Real data will have many attributes in addition to the attributes we are using as skyline criteria. Also, attributes suitable as skyline conditions are comparables, as integer, float, and date. These tend to be small, storage-wise. A tuple's other attributes will likely include character data and be relatively large, storage-wise. So projecting out the non-skyline attributes of tuples when we add them to the window can be a great benefit. Significantly more skyline tuples will fit into the same size window. Likewise, there is no need to ever keep duplicate (projected) tuples in the window. So we can do duplicate elimination, which also makes better use of the window space.

Dimensional Reduction

Another optimization available to **SFS** is due again to the fact that we first sort the table. Recall the nested sort that results from the query in Figure 6. Now consider the table that results from the query in Figure 7. It has precisely the same skyline as table $\mathbf{R}$. We choose the maximum a_k for each "$a_1, \ldots, a_{k-1}$" group. Clearly, any tuple in the group but with a non-maximum a_k cannot belong to the skyline. Of course, we can only apply this reduction once. (Implemented internally, other attributes of $\mathbf{R}$ besides the a_i's could be preserved during the "`group by`" computation.)

This optimization is useful in cases when the number of distinct values for each of the attributes $a_1, \ldots, a_{k-1}$ appearing in $\mathbf{R}$ is small, so that the number of groups is much smaller than the number of tuples. If one attribute

select $a_1, \ldots, a_{k-1}, max(a_k)$ *as* a_k from R
group by $a_1, \ldots, a_{k-1}$;
order by a_1 *desc*, $\ldots, a_{k-1}$ *desc*;

Fig. 7. An order by query that produces a total monotone order.

has many distinct values, we can make this one our "a_k". In such a case, we are applying **SFS** to the result of the query in Figure 7, which can be a much smaller input table.

4 Conclusions

We believe that the skyline operator offers a good start to providing the functionality of preference queries in relational databases, and would be easy for users to employ. We believe that our **SFS** algorithm for skyline offers a good start to incorporating the skyline operator into relational engines, and hence, into the relational repertoire, effectively and efficiently.

References

1. R. Agrawal and E. L. Wimmers. A framework for expressing and combining preferences. In *Proceedings of SIGMOD*, pages 297–306, 2000.
2. 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. *JACM*, 25(4):536–543, 1978.
3. S. Börzsönyi, D. Kossmann, and K. Stocker. The skyline operator. In *Proceedings of ICDE*, pages 421–430, 2001.
4. J. Chomicki. Querying with intrinsic preferences. In *Proceedings of EDBT*, 2002.
5. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. Technical Report CS-2002-04, Computer Science, York University, Toronto, Ontario, Canada, Oct. 2002.
6. J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with presorting. In *Proceedings ICDE*, pages 717–719, 2003.
7. P. Godfrey. Cardinality estimation of skyline queries: Harmonics in data. In *Proceedings of FoIKS Conference*, pages 78–97, 2002.
8. V. Hristidis, N. Koudas, and Y. Papakonstantinou. PREFER: A system for the efficient execution of multi-parametric ranked queries. In *Proceedings of SIGMOD*, pages 259–270, 2001.
9. W. Kiessling. Foundations of preferences in database systems. In *Proceedings of the 28*th *VLDB*, Aug. 2002.
10. H. T. Kung, F. Luccio, and F. P. Preparata. On finding the maxima of a set of vectors. *JACM*, 22(4):469–476, 1975.
11. F. P. Preparata and M. I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 1985.
12. K.-L. Tan, P.-K. Eng, and B. C. Ooi. Efficient progressive skyline computation. In *Proceedings of VLDB*, pages 301–310, 2001.