Skyline Queries, Front and Back*

Jan Chomicki
University at Buffalo
Buffalo, NY 14260
chomicki@buffalo.edu

Paolo Ciaccia
University of Bologna
40136 Bologna, Italy
paolo.ciaccia@unibo.it

Niccolo' Meneghetti
University at Buffalo
Buffalo, NY 14260
niccolom@buffalo.edu

ABSTRACT

Skyline queries are a popular way to obtain preferred answers from the database by providing only the orderings of attribute values. The result of a skyline query consists of those input tuples for which there is no input tuple having better or equal values in all the attributes and a better value in at least one attribute. In this article, we summarize the basic notions and properties of skyline queries, and discuss their extensions and generalizations. In particular, we consider skyline algorithms and skyline cardinality issues.

1. INTRODUCTION

Multi-criteria analysis [16] is a common approach to address the needs of decision-making applications. In some such settings a set of dimensions and a set of alternatives, both finite, are given. For example, a car shopper considers the price, make, model and mileage of the car, as well as the vehicles currently in stock. The analysis identifies the best, *most preferred* alternatives, which are obtained by eliminating those that are *dominated* by other alternatives. Under *Pareto* efficiency, the dominance relationship has a particularly simple structure: an alternative o_1 dominates an alternative o_2 if o_1 is better than or equal to o_2 in all dimensions and better than o_2 in at least one dimension. (An equivalent formulation requires that o_1 and o_2 be different, and o_1 be better than or equal to o_2 in all dimensions.)

In the database context, the concepts of multi-criteria decision analysis have a very natural interpretation. The dimensions correspond to attributes, and the alternatives, to the objects present in the database. Pareto dominance leads to *skyline* queries.

EXAMPLE 1.1. *A prospective student, choosing a school, takes into account the financial support promised by the school, as well as its percentile rank among all schools. We represent the relevant data using a relation* `School(Id,Rank,Support)`*, which is depicted in Table 1 and plotted in Figure 1.*

id	rank	support	id	rank	support
t_1	96	5,000	t_8	96	3,000
t_2	95	6,000	t_9	93	3,500
t_3	89	8,000	t_{10}	92	2,500
t_4	87	9,000	t_{11}	88	4,500
t_5	86	10,000	t_{12}	85	7,000
t_6	84	14,000	t_{13}	83	6,500
t_7	81	14,500	t_{14}	80	11,000

Table 1: Student database

Suppose the student does not have a scoring function that assigns a numeric score to each school: such functions are difficult to construct. However, the student can still determine whether a school is dominated by some other school providing better support and having the same or higher rank, or having a higher rank and providing the same or better support. The dominated schools represent inferior choices and can be eliminated. The remaining schools are nondominated and form the skyline *of the input. The skyline contains all the best choices, in a precise sense. In this case, the skyline consists of the tuples* $t_1, t_2, t_3, t_4, t_5, t_6$*, and* t_7 *(the black dots).*

The notion of skyline queries was pioneered in [8]. Subsequently, the interest in this area has exploded: [8] has garnered over 1200 citations (Google Scholar, May 2013). The research has uncovered interesting properties of skylines, designed efficient algorithms for computing skyline queries, and produced numerous generalizations and extensions of the basic framework. Unfortunately, the sheer volume of publications on skylines and related topics prevents us from covering here all interesting issues

*Database Principles Column. Column editor: Pablo Barceló, Department of Computer Science, Universidad de Chile, Santiago, Chile. E-mail: pbarcelo@dcc.uchile.cl.

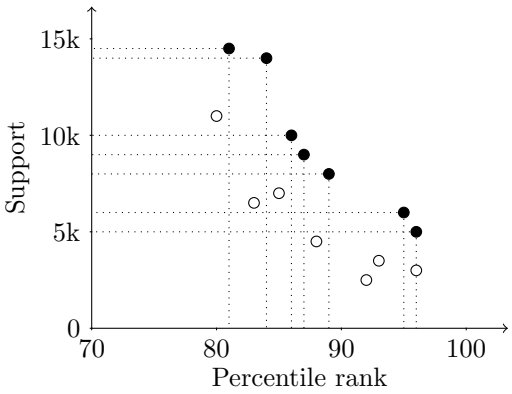

Figure 1: Student database (plotted)

studied in this area.

The needs of decision-making applications have been calling for designing new classes of queries that go beyond skyline queries. Richer classes of preferences and forms of query results have been studied. The research is motivated not only by semantic concerns but also by the observation, confirmed both analytically and experimentally, that skyline sizes may get impractically large.

It is important to keep in mind the distinctions between (1) Pareto dominance, (2) skyline queries that return all the non-Pareto-dominated objects, (3) skylines: the results of skyline queries, and (4) skyline algorithms that compute skylines.

The plan of the paper is as follows. In Section 2, we introduce the basic notions of Pareto dominance and study its basic properties. In Section 3, we consider skyline queries and their variants. Section 4 contains a survey of skyline algorithms. In Section 5, we address the issue of skyline cardinality. Section 6 shows generalizations and adaptations of Pareto dominance to various contexts. In Section 7, we consider some issues that we believe will be of importance in future research on skylines: uncertainty and elicitation of dominance relations. We conclude in Section 8.

2. DOMINANCE RELATIONS

2.1 Pareto dominance

We present here first a basic formal model of *Pareto dominance* (also called *Pareto preference*). Later, we will show how Pareto dominance can be generalized and extended.

Let $\mathcal{A} = \{A_1, ..., A_d\}$ be a finite set of attributes (a relation schema). The number d is the *dimension* of Pareto dominance. Every attribute $A_i \in \mathcal{A}$ is associated with an *infinite domain* $\mathcal{D}_{A_i}$ (here: real numbers). We work with the *universe of tuples* $\mathcal{U} = \prod_{A_i \in \mathcal{A}} \mathcal{D}_{A_i}$. A *dominance* (preference) relation is a subset of $\mathcal{U} \times \mathcal{U}$. Given a tuple $t \in \mathcal{U}$, we denote the value of its attribute A_i by $t[A_i]$. This notation can be easily generalized to sets of attributes.

Every attribute $A_i \in \mathcal{A}$ is associated with one of the standard orderings of the reals, $>$ or $<$, denoted by $>_{A_i}$. For simplicity, we assume in this paper that $>_{A_i}$ is $>$ (*larger is better*). The Pareto dominance relation $\succ^{pto}$ is defined as

$$t \succ^{pto} s \equiv t \neq s \wedge \bigwedge_{A_i \in \mathcal{A}} t[A_i] \geq_{A_i} s[A_i],$$

or, equivalently, as

$$t \succ^{pto} s \equiv \bigvee_{A_i \in \mathcal{A}} t[A_i] >_{A_i} s[A_i] \wedge \bigwedge_{A_i \in \mathcal{A}} t[A_i] \geq_{A_i} s[A_i].$$

These are *strict* versions; there is also a *non-strict* version obtained in the standard way:

$$t \succeq^{pto} s \equiv \bigwedge_{A_i \in \mathcal{A}} t[A_i] \geq_{A_i} s[A_i].$$

Strictly speaking, $\succ^{pto}$ denotes a different relation for different schemas $\mathcal{A}$. In the cases where it is necessary to disambiguate the set of attributes over which dominance is defined, we will add an appropriate subscript. In practical applications, some relation attributes may be irrelevant for Pareto dominance: see the attribute *id* in Example 1.1. For simplicity, we do not consider such attributes.

From an algebraic perspective, Pareto dominance can be defined using a binary accumulation operator $\otimes$ (also called Pareto) combining the attribute orders $>_{A_i}, i = 1, \ldots, d$:

$$\succ^{pto}_{\mathcal{A}} = >_{A_1} \otimes >_{A_2} \otimes \cdots \otimes >_{A_d}$$

where $\succ^{pto}_{A_i} = >_{A_i}$ and $\succ^{pto}_{XY} = \succ^{pto}_{X} \otimes \succ^{pto}_{Y}$ is defined as

$$t[XY] \succ^{pto}_{XY} s[XY] \equiv t[X] \succ^{pto}_{X} s[X] \wedge t[Y] \succeq^{pto}_{Y} s[Y] \vee t[X] \succeq^{pto}_{X} s[X] \wedge t[Y] \succ^{pto}_{Y} s[Y]$$

for $XY \subseteq \mathcal{A}$ and $X \cap Y = \emptyset$.

Pareto accumulation is associative and commutative.

2.2 Properties

Representation. Clearly, $\succ^{pto}$ is irreflexive and transitive. However, it is not negatively transitive (its complement does not have to be transitive), and thus $\succ^{pto}$ fails to be a weak order, as in the example below:

$(2,0) \not\succ^{pto} (0,2), (0,2) \not\succ^{pto} (1,0), (2,0) \succ^{pto} (1,0)$.

The lack of the weak order property implies that $\succ^{pto}$ *cannot be represented using any scoring function.* Formally, a scoring function f represents the dominance relation $\succ_f$ such that

$$t \succ_f s \equiv f(t) > f(s).$$

To show that $\succ_f$ is a weak order, suppose $t \succ_f s$. Then $f(t) > f(s)$. So for every w, $f(t) > f(w)$ or $f(w) > f(s)$, and thus $t \succ_f w$ or $w \succ_f s$. Therefore, $t \not\succ_f w$ and $w \not\succ_f s$ imply $t \not\succ_f s$.

Robustness. One way of characterizing the robustness of $\succ^{pto}$ is through *invariance* with respect to the transformations of the underlying space. A function g is *Pareto-invariant* if for all tuples t and s, $t \succ^{pto} s$ implies $g(t) \succ^{pto} g(s)$.

A useful related notion is that of *monotonicity.* A function f mapping $\mathcal{U}$ into some Cartesian product of domains $\mathcal{V}$ is *monotone* if for all tuples t and s, $t \succeq^{pto} s$ implies $f(t) \succeq^{pto} f(s)$.

One can define special classes of monotone functions, for example *shifting* (addition of a vector of constants) and *scaling* (multiplication by a vector of constants). Shifting is Pareto-invariant. Scaling by an all-positive vector is also Pareto-invariant. However, scaling by a vector that has a zero component is not necessarily Pareto-invariant, as Example 2.1 shows. So monotonicity does not imply Pareto-invariance.

EXAMPLE 2.1. *Consider the tuples* $t = (3,1)$ *and* $s = (2,1)$*, and the scaling vector* $(c_1, c_2) = (0,1)$*. Clearly,* $t \succ^{pto} s$ *but*

$$(c_1 \cdot t[A_1], c_2 \cdot t[A_2]) \not\succ^{pto} (c_1 \cdot s[A_1], c_2 \cdot s[A_2]).$$

3. SKYLINE QUERIES

3.1 Skyline using winnow

There is more than one way to incorporate the notion of Pareto dominance into a relational query language. We call all such queries *Pareto queries.* Arguably, the simplest and the most widely used kind of a Pareto query is the relational operator with the meaning *"Retrieve all the nondominated tuples in the input relation."* We use the term *winnow* for this operator, and the notation $\omega_\succ(R)$ where $\succ$ is a dominance relation and R a database relation schema [10]. Typically $\succ$ is defined by a logic formula. The semantics of winnow is as follows:

$$\omega_\succ(r) = \{t \in r \mid \neg\exists t' \in r.\ t' \succ t\}$$

where r is an instance of R. A *skyline query* is a special instance of winnow in which $\succ$ is $\succ^{pto}$, and is written as $sky_{\mathcal{A}}(R)$ (where $\mathcal{A}$ is the set of all attributes of R). The result $sky_{\mathcal{A}}(r)$ of a skyline query for an instance r of R is called a *skyline*: $sky_{\mathcal{A}}(r) = \omega_{\succ^{pto}}(r)$. The size of the skyline is denoted by $\ell = |sky_{\mathcal{A}}(r)|$. Intuitively, the Pareto-dominated tuples are inferior and should not be of interest to the user.

3.2 Maxima of scoring functions

As Theorem 3.1 below shows, a skyline consists of the maxima of monotone scoring functions. Thus, even if the scoring function is not known, the highest-scoring tuple is guaranteed to be in the skyline. This result was already mentioned in the original skyline paper [8]. A proof appeared in the full version of [13]. We present a different proof here.

THEOREM 3.1.

$$\forall r.\ \forall t \in r.\ t \in \mathrm{sky}_A(r) \text{ iff } \exists f \in \mathcal{M}.\ \forall t' \in r.\ t' \neq t \Rightarrow f(t) > f(t'),$$

where $\mathcal{M}$ *is the set of all monotone scalar-valued functions over* $\mathcal{U}$*.*

PROOF. $\Leftarrow$ Assume RHS holds and $t \notin sky_A(r)$. Then there is $t' \neq t$ such that $t' \succ^{pto} t$. By the monotonicity of f, $f(t') \geq f(t)$. A contradiction.

$\Rightarrow$ Assume $t \in sky_A(r)$. Without lack of generality, we can assume all attribute values are strictly positive: a simple shifting operation can make all the tuples strictly positive; the composition of a translation with a monotone function is again a monotone function. It is easy to see that for every $t' \neq t$, there is an attribute A_i such that $t[A_i] > t'[A_i]$ (otherwise t would not be in the skyline). The function f is defined for every t' as

$$f(t') = \min_i \{\frac{t'[A_i]}{t[A_i]} \mid i = 1, \ldots, d\}.$$

Thus $f(t) = 1$ and $f(t') < 1$ for $t' \neq t$. Also, f is monotone. □

3.3 Algebraic laws

Algebraic laws relating query operators serve as a foundation of query optimization. Such laws can be used as rewrite rules to produce different, presumably more efficient formulations of queries. One of the basic insights of this field is that it is useful to push low-cost, size-reducing operators like relational algebra selection below high-cost, size-increasing operators like join. Such a transformation usually reduces the size of intermediate query results and the overall cost of evaluating the query.

The above intuition applies to skyline queries as well. A skyline query (winnow) is a high-cost operation, so it pays to push a selection below it.

THEOREM 3.2. *[10] If for every t and s such that $t \succ^{pto} s$ $\alpha(s)$ implies $\alpha(t)$, then for every relation instance r*

$$\sigma_\alpha(\text{sky}_\mathcal{A}(r)) = \text{sky}_\mathcal{A}(\sigma_\alpha(r)).$$

In Example 1.1, the selection $\sigma_{Rank>80}$ commutes with the skyline query but the queries $\sigma_{Rank<80}$ and $\sigma_{Rank=80}$ do not.

Other algebraic laws involving skyline and more general queries are studied in [10, 11, 26, 31].

3.4 Subspace skylines

In addition to the full-space skyline $sky_A(R)$, subspace skylines $sky_\mathcal{B}(R)$ for $\mathcal{B} \subset \mathcal{A}$ [17] are also considered in order to extract more information from the input dataset. For example, subspace skylines show which points excel in specific dimensions and which have all-around strength.

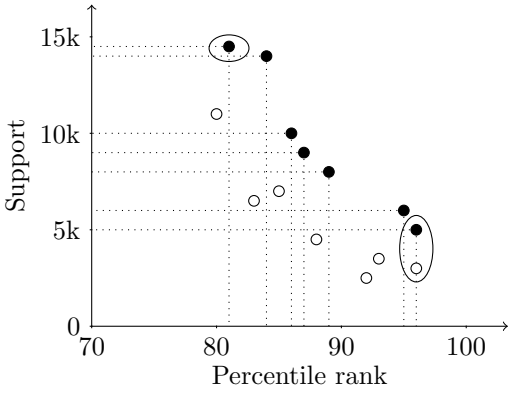

Figure 2: Subspace skylines

EXAMPLE 3.1. *We use the setting of Example 1.1. In Figure 2, the two one-dimensional subspace skylines are circled with ovals, and the two-dimensional skyline elements are black. Interestingly, a subspace skyline does not have to be contained in higher-dimensional skylines, unless no attribute values can be shared by different objects [17].*

Subspace skylines lead to new, interesting classes of queries. The *skyline membership query* determines the subspaces where a given object is in the subspace skyline. The *skycube query* returns the skylines in all the nonempty subspaces.

3.5 Other query classes

It is remarkable that Pareto dominance, a single kind of dominance relation, has been used to define many new classes of queries [41]. We briefly survey some of those classes here. *Constrained skyline queries* are skyline queries applied to the result of a selection. *Ranked skyline queries* are skyline queries whose output is ranked using a scoring function. When the scoring function returns the number of objects dominated by the given object, we have *enumerating queries*; when only top-k objects are returned, the queries are *k-dominating.* Finally, *k-skyband queries* return the objects dominated by at most k other objects. Clearly, skyline queries are 0-skyband.

4. SKYLINE ALGORITHMS

The skyline of a relation instance r with n tuples and attributes $\mathcal{A} = \{A_1, ..., A_d\}$ can be naïvely computed in $\mathcal{O}(n^2 d)$ time, which is $\mathcal{O}(n^2)$ if the relation schema is fixed. The quadratic bound holds since a standard nested-loops (NL) join algorithm, which compares a tuple t to each other tuple s, clearly suffices to determine if $t \in sky_\mathcal{A}(r)$.

The NL algorithm always requires $\Theta(n^2)$ time. This is also the worst-case performance of any algorithm that is applicable to arbitrary dominance relations $\succ$ and oblivious of the d-dimensional Euclidean space in which the tuples lie (as NL indeed is): If $\forall t, s \in r$ it is the case that $t \not\succ s$ and $s \not\succ t$, thus $\omega_\succ(r) = r$, then all pairs of tuples will be compared. Based on this observation, it can be argued that there are (at least) two features that can be used to escape the quadratic lower bound (or at least to reduce costs on the average) and that NL ignores: transitivity of the dominance relation and the attribute orders $>_{A_i}, i = 1, \ldots, d$. Further, it is advisable to look at *output-sensitive* algorithms, for which the performance depends on the skyline size, since the case of small-to-medium skylines is the most interesting one from a practical viewpoint.

In the following we provide a concise picture of the major approaches adopting such ideas and others as well. It is important to bear in mind that the problem of computing the skyline is equivalent to the *maximal vectors* problem studied in computational geometry, for which several algorithms are known. Here we do not cover them in detail, both for space reasons and for the absence of experimental validation on large databases.

4.1 Using transitivity

BNL. The BNL (block nested-loops) algorithm [8] is a practical way of implementing the NL approach in a database system. BNL allocates a buffer (window) W in main memory, and sequentially scans r. For each newly read tuple t, it compares t to the tuples in W. If $\exists s \in W.s \succ^{pto} t$, then t is discarded, otherwise t is inserted into W and the tuples in W dominated by t are removed. In case W saturates,

a temporary file F is used to store those tuples that overflow. At the end of this process, all tuples that were inserted in W when F was still empty can be output. Another pass then starts in which F is used as the input, and the process repeats until no overflow occurs.

Empirical evaluation of BNL shows its high dependency on how the tuples are distributed, the worst case being when they are negatively correlated. BNL effectiveness also largely depends on the order in which the tuples are processed and on the window size. Indeed, for an unlimited window size, $|W| \geq n$, BNL might still require $\Theta(n^2)$ time *regardless of the skyline size*. This can be seen by considering the case in which tuples are ordered as $\langle t_1, t_2, \ldots, t_n \rangle$, $t_n \succ^{pto} t_i, i = 1, \ldots, n-1$, and no other dominance relationship exists (thus $sky_{\mathcal{A}}(r) = \{t_n\}$). At the other extreme, $|W| = 1$ guarantees that at most $\mathcal{O}(\ell n)$ comparisons will be performed, where $\ell = |sky_{\mathcal{A}}(r)|$. However the actual running time might be negatively influenced by having a small window, since I/O costs will increase. Notice that, although BNL has been empirically analyzed and its performance contrasted to that of several other algorithms, a complete understanding of this algorithm's behavior is still lacking. For instance, the scenario in which a large amount of space in main memory is available *and* a small window is used, with the rest of available memory used to cache F, has not been considered yet.

SFS. The SFS (Sort-Filter-Skyline) algorithm [12, 13] is similar to BNL, but it first topologically sorts r using a monotone function f. In the resulting order $\langle t_1, t_2, \ldots, t_n \rangle$ it is therefore guaranteed that $i < j \Rightarrow t_j \not\succ^{pto} t_i$. This property leads to three major improvements with respect to BNL: 1) SFS is *progressive*, since $t \in W$ implies $t \in sky_{\mathcal{A}}(r)$, and therefore t can be immediately output; 2) the number of passes is optimal, $\lceil |\ell|/|W| \rceil$; and, 3) if t and s are both dominated tuples, then they will not be compared to each other. As a consequence of 3), SFS will not execute more than $\mathcal{O}(\ell n + n \log n)$ comparisons ($n \log n$ being paid for sorting r).

LESS and SaLSa. There are two orthogonal directions along which SFS can be improved, and these yield the LESS and SaLSa algorithms, respectively. The basic idea of LESS [23] is to anticipate the dominance tests in the sorting phase, so as to discard some dominated tuples earlier, and consequently to reduce the sorting costs. SaLSa [4] extends SFS by avoiding to read the whole sorted input relation. Let $\langle t_1, t_2, \ldots, t_n \rangle$ be the order in which tuples are read, with t_i $(i < n)$ being the last fetched tuple. Since the function f used to sort tuples is monotone, it is $\forall j > i . f(t_i) \geq f(t_j)$, thus all unread tuples correspond to points in a bounded region BR. (This requires that the attribute domains be bounded from below, which is always true if one considers *active* domains.) Therefore, if there exists t_j $(j \leq i)$ such that t_j Pareto-dominates BR, that is, $\forall s \in BR . t_j \succ^{pto} s$, then the algorithm can be halted since no more tuples will enter the skyline. Theoretical analysis reveals an interesting fact about the *limiting* capability of SaLSa.

THEOREM 4.1. *[4] Let $m \leq n$ be the number of tuples that SaLSa reads. For any data distribution, the expected value of m/n monotonically decreases with n.*

Remark: Although none of the described algorithms can avoid a quadratic cost in the worst case in which the skyline has size $\Theta(n)$, their *average-case* behavior is indeed much better, as also confirmed by experimental observations and analytical results. For instance, the analysis in [24] proves that LESS has a *linear*, $\mathcal{O}(dn)$, complexity under the assumptions of independence of attributes, uniform distribution, and low probability of duplicate attribute values. A similar result is derived for BNL with either unlimited ($|W| \geq n$) or minimal ($|W| = 1$) window size, whereas for SFS it is shown that sorting is, in terms of average performance, equivalent to reducing the skyline dimensionality by one. Unfortunately, the simplifying assumptions on which results like these are based rarely hold together in real databases.

4.2 Using attribute orders

Since attribute domains are totally ordered, it is possible to partition them. This idea is at the heart of *divide & conquer* approaches, which have been pioneered in the computational geometry field. We first detail (Algorithm 1) the basic scheme of Kung et al. [33] and then discuss the D&C algorithm by Börzsönyi et al. [8] that was developed for dealing with large instances that do not fit in main memory.

After partitioning on 2 attributes A_i and A_j, the sets $S_{H_i,H_j}, S_{H_i,L_j}, S_{L_i,H_j}$, and S_{L_i,L_j} are obtained, with sets S_{H_i,L_j} and S_{L_i,H_j} that do not need to be compared. This observation, together with a rather sophisticated merging scheme, leads to the worst-case subquadratic bound $\mathcal{O}(n \log^{d-2} n)$ $(d \geq 3)$. For $d = 2, 3$ this reduces to $\mathcal{O}(n \log n)$, which meets the theoretical lower bound established in [33].

The D&C algorithm shares with the above scheme the idea of recursive partitioning, but at each step it generates an *m-way partition* (rather than a 2-way one as in [33]), where m is chosen so that each

Algorithm 1 Basic divide & conquer [33]

Input: instance r with schema $\mathcal{A} = \{A_1, \ldots, A_d\}$
Output: $sky_{\mathcal{A}}(r)$

1: Partition r using the median m_i of some attribute A_i. Let $S_{H_i} = \{t \in r.t_i \geq m_i\}$ and $S_{L_i} = r \setminus S_{H_i}$;
2: Compute $sky_{\mathcal{A}}(S_{H_i})$ and $sky_{\mathcal{A}}(S_{L_i})$ by recursively applying step 1;
3: Merge $sky_{\mathcal{A}}(S_{H_i})$ and $sky_{\mathcal{A}}(S_{L_i})$, i.e., determine $T_{L_i} \subseteq sky_{\mathcal{A}}(S_{L_i})$ s.t. $sky_{\mathcal{A}}(r) = sky_{\mathcal{A}}(S_{H_i}) \cup T_{L_i}$.

of the resulting sets can be loaded in main memory. Although this makes D&C more amenable to a database scenario, its simplified merging scheme causes the worst-case complexity to rise back to $\mathcal{O}(n^2)$.

If one considers an external memory (EM) model, in which CPU is free and the cost is measured in terms of I/O operations, the currently best result is due to Sheng and Tao [45]. By developing a smart m-way merging technique, they are able to compute the skyline with $\mathcal{O}(n/B \log^{d-2}_{M/B}(n/B))$ I/Os ($d \geq 3$), where B (M) is the number of tuples in each disk block (main memory, respectively).

Remark: Although divide & conquer algorithms typically exhibit a subquadratic worst-case complexity, this does not imply their superiority over other approaches in terms of actual running time, to which many other factors contribute, for example the hidden constant factors in $\mathcal{O}()$ notation.

4.3 Low-cardinality domains

In many situations, some (or even all) attributes of interest can only assume values from a limited set of alternatives, i.e., domains have low cardinality. For instance, ratings of movies and hotels typically are integers in a small range, say $[1, 5]$. Also, *Boolean attributes* are typically used to describe the presence/absence of interesting object features.

Without loss of generality, we assume here that each combination of skyline attribute values can occur multiple times and $\forall A_i \in \mathcal{A}. |\mathcal{D}(A_i)| = V \ll n$. The LS-B algorithm [39] first builds the complete lattice of all the V^d value combinations ordered by Pareto dominance, and marks all elements as *not present* (`np`). It then sequentially reads the input relation r, and for each tuple t it marks as *present* (`p`) the corresponding lattice element. A simple level-wise analysis of the lattice is executed to determine which are the `p`-values that are also nondominated, and that consequently are in the skyline. Finally, with a second scan of r all skyline tuples are computed. (It is needed since tuples may have other attributes besides those on which the skyline is computed.) Overall, LS runs in $\mathcal{O}(dV^d + dn)$ time, which reduces to $\mathcal{O}(dn)$ if $V = \mathcal{O}(n^{1/d})$.

LS-B can also be adapted to work when (exactly) one attribute, say A_d, has *not* a low-cardinality domain. The idea of the extended algorithm, called LS, is to store in each lattice element also the *locally optimal value* (`lov`) of A_d for that element. Since $\succ^{pto}_{\mathcal{A}} = \succ^{pto}_{B} \otimes \succ^{pto}_{A_d}$, where $B = \mathcal{A} \setminus \{A_d\}$ and $\succ^{pto}_{A_d} = >_{A_d}$, for distinct tuples t and s it is:

$$t \succ^{pto}_{\mathcal{A}} s \equiv t[B] \succeq_B s[B] \wedge t[A_d] \geq_{A_d} s[A_d].$$

Consequently, a tuple s can be discarded if: (1) there exists a lattice element marked `p` that dominates the element of s and whose `lov` is at least as high as $s[A_d]$; or (2) $s[A_d]$ is strictly less than the `lov` of its element (implying that $\exists t.t[B] = s[B] \wedge t[A_d] >_{A_d} s[A_d]$). It is simple to show that the complexity of LS is the same as that of LS-B.

Unfortunately, no simple extension to the general case in which a mix of low- and high-cardinality domains coexist seems to be possible. Indeed, if two attributes have high cardinality, the *local skyline* with respect to these attributes should be computed for each lattice element, a fact that might nullify the advantages of using a lattice-based approach.

4.4 Index-based approaches

As with any other query type, processing of skyline queries can be accelerated if the input data have been indexed.

The BBS (Branch-and-Bound Skyline) algorithm [40] assumes that r is indexed by an R-tree, for which index regions are minimum bounding rectangles (MBRs), and that a *target (reference) point* p is available. Without loss of generality, let p be any point such that $\forall t.p[A_i] \geq t[A_i]$, $i = 1, \ldots, d$, so that the assumption *larger is better* still holds.

BBS performs an ordered scan of the index nodes based on their L_1 distance from p, i.e., for a node N whose region is $Reg(N) = [l_1, h_1] \times \ldots \times [l_d, h_d]$, it is $L_1(Reg(N), p) = \sum_i |p[A_i] - h_i|$. Notice that for each point $t \in Reg(N)$ it is guaranteed that $L_1(Reg(N), p) \leq L_1(t, p)$. The region descriptions of the nondominated index nodes that have not been accessed yet, as well as the currently nondominated points retrieved so far, are organized together in a priority queue PQ, which is kept ordered by increasing values of L_1 distances. Notice that a point t Pareto-dominates a node N, $t \succ^{pto} N$, if $\forall i.t[A_i] \geq h_i$. (If a bag semantics is assumed, then at least one inequality needs to be strict.)

Since L_1 (as well as any other L_p norm) is a monotone function, as soon as a point t becomes the top

element of PQ it is guaranteed that t belongs to the skyline. Thus, BBS is *progressive*. Monotonicity of L_1 is also the key to show that BBS is *I/O-optimal*, that is, only the index nodes for which inspection of the points they contain is necessary to ensure the correctness of the result are accessed.

An index-based solution based on the *Z-order*, which maps multidimensional points to a linear address space, is introduced in [34]. The proposed 1-dimensional index structure, called ZBtree, is based on the B$^+$-tree principles, in that each node region is a 1-dimensional interval, i.e., a sequence of *Z-addresses*, and intervals do not overlap. Peculiar to the ZBtree is the policy according to which points and region descriptions are packed together (in leaf and non-leaf nodes, respectively), and which aims to facilitate the pruning of some regions, thus avoiding dominance tests.

4.5 Distribution and parallelism

Vertical fragmentation. Consider a scenario in which the d skyline attributes are distributed over multiple sites, each site thus providing only a partial view of the alternatives under examination. In the following we describe the basic case in which each site stores a single skyline attribute (thus, there are exactly d sites), the extension to arbitrary vertical decompositions having been recently analyzed in [48].

The BDS algorithm [3] is based on the framework that Fagin pioneered for the processing of top-k ranking queries [18, 19], and that since then has been widely used for retrieving data from multiple sources. According to Fagin's framework there are d sorted lists $L_i, i = 1, \ldots, d$, with the i-th list ordered by non-increasing values of attribute A_i, and all lists managing the same set of n objects $O_1, \ldots, O_n$. A d-way 1-1 join on the object identifiers then allows the instance r to be reconstructed. Lists can be accessed either requesting the next element in the order (*sorted access*) or by providing an identifier and requesting the associated attribute value (*random access*). The basic steps of BDS are summarized in Algorithm 2.

The condition that halts the first phase is based on monotonicity (if an object O has not been seen in any list, then it is $\forall i. O_s[A_i] \geq_{A_i} O[A_i]$, thus $O_s \succ^{pto} O$), and is already found in [8]. There, the above algorithm is considered for computing the skyline using d B$^+$-trees, i.e., a different scenario but with the same access model of BDS.

Example 4.1. *Consider the example in Table 2, in which there are $d = 3$ sorted lists and* $\text{sky}_{\mathcal{A}}(r) = \{O_2, O_7\}$. *After retrieving the first 3 objects from each list BDS can halt, since all unseen objects, like O_1, are dominated by O_2.*

Algorithm 2 Basic distributed skyline algorithm

Input: instance r vertically partitioned in d sorted list $L_1, \ldots, L_d$
Output: $sky_{\mathcal{A}}(r)$
1: Cyclically perform sorted accesses on the d lists until (at least) one object, say O_s, is retrieved from *all* the lists;
2: For all objects O that have been fetched from *at least one* list, perform random accesses to retrieve the missing attribute values;
3: Perform the necessary dominance tests and return the nondominated objects.

oid	A_1
O_7	0.9
O_3	0.6
O_2	0.6
O_1	0.5
O_4	0.4

oid	A_2
O_2	0.9
O_3	0.7
O_4	0.6
O_1	0.5
O_7	0.5

oid	A_3
O_7	1.0
O_2	0.8
O_4	0.7
O_3	0.7
O_1	0.6

Table 2: A vertically-partitioned relation

The halt condition coincides with the one used in the A_O algorithm by Fagin [18] for computing the top-1 object according to an arbitrary monotone scoring function. In light of Theorem 3.1, this should not be surprising at all. This fact also implies that the analysis in [18] applies to skyline computation, which with arbitrarily high probability, and assuming attribute independence, will therefore require $\mathcal{O}(n^{1-1/d})$ accesses to the lists for any fixed value of d.

Horizontal fragmentation. When a relation r is horizontally fragmented over a cluster of P servers, $r = r_1 \cup \ldots \cup r_P$, the skyline can be computed by exploiting the identity

$$sky_{\mathcal{A}}(r) = sky_{\mathcal{A}}(sky_{\mathcal{A}}(r_1) \cup \ldots \cup sky_{\mathcal{A}}(r_p)),$$

i.e., by first computing the *local* skylines and then merging the results, as the divide & conquer approaches described in Section 4.2 do. As argued in [1], this simple scheme has the drawback of requiring $\log P$ communication steps, i.e., its *synchronization complexity* is high and might easily become the main performance bottleneck. The (first) algorithm described in [1] requires only 2 communication steps and is also perfectly load-balanced, in that each server has maximum load $\mathcal{O}(dn/P)$.

The algorithm starts from an arbitrary data allocation, in which each server s manages a local

fragment r_s of n/P tuples. In the initial preprocessing phase the servers cooperate to build a grid of P^d cells, which requires a total of $dP(P+1)$ values (partition points) to be transmitted over the network (note that this is independent of n). A key property of the multidimensional grid, in which each coordinate is partitioned into P buckets, is that each bucket is guaranteed to contain $\mathcal{O}(n/P)$ points. Assuming J is the set of nonempty cells, the preprocessing phase also includes the broadcast of which cells are in J (there are P^{d+1} values overall). Knowing J immediately allows the points within cells that are *strictly dominated* by some other cell in J to be discarded, where cell C strictly dominates cell D if on each coordinate C has a value strictly better than D. All cells in J that are not strictly dominated form the so-called *relaxed skyline* of r, $S_r(J)$, and they are guaranteed to contain all the points in $sky_{\mathcal{A}}(r)$.

If C is a cell in the relaxed skyline, and $t \in C$, then t can only be dominated by points in the cells of $S_r(J)$ that dominate C. A key observation is that these are exactly those cells D that share with C at least one coordinate value (thus, on at least one dimension they belong to the same bucket), and have better values in the other coordinates. Based on this observation, it is derived that the total number of points that can dominate any point t in a cell C is $\mathcal{O}(dn/P) = \mathcal{O}(n/P)$, i.e., an amount of data that could be processed by a single server. However, since the number of cells in the relaxed skyline can be in the order of $\mathcal{O}(P^{d-1})$, it is unfeasible to look for skyline points on a cell-by-cell basis.

The first step of the algorithm in [1] assigns to each server s the task of computing the local skylines $sky_{\mathcal{A}}(r_{i,s})$, $i = 1, \ldots, d$, where $r_{i,s} = \{t \in r | t \in C = (C_1, \ldots, C_d) \wedge C \in S_r(J) \wedge C_i = s\}$ is the set of points mapped to those cells C of the relaxed skyline that on the i-th coordinate fall in bucket s. The identity

$$r_1 \cap r_2 \cap sky_{\mathcal{A}}(r_1 \cup r_2) = sky_{\mathcal{A}}(r_1) \cap sky_{\mathcal{A}}(r_2)$$

is then used to compute the result, the intuition being that if $t \in C$, then $t \in sky_{\mathcal{A}}(r)$ iff it is in the local skylines $sky_{\mathcal{A}}(r_{i,C_i})$, $i = 1, \ldots, d$. The second step of the algorithm computes such intersections using a randomized load-balanced algorithm.

4.6 Further approaches

Several other algorithms have been developed for computing the skyline in distributed environments. In particular, both structured and unstructured peer-to-peer networks have been considered and specific techniques for processing skyline queries in such scenarios have been developed. A recent survey [27] enters into the details of these approaches.

5. SKYLINE CARDINALITY

5.1 Average skyline size

Estimating the expected skyline size for a given data distribution is a key issue for the design of good cost models for skyline queries. The problem has been addressed in several papers [7, 9, 22]; here we report some major results.

Denote by $\ell_{d,n}$ the expected size of the skyline for a randomly sampled dataset of n points in d dimensions. Under the assumption that (i) all the attributes are statistically independent of each others, and (ii) the probability of sampling the same attribute value twice is negligible, the following recurrence holds:

$$\ell_{d,n} = \ell_{d,n-1} + \frac{1}{n} \cdot \ell_{d-1,n} \tag{1}$$

where the base case is $\ell_{d,1} = 1$.
If we assume that any two different tuples in r cannot share the same value for the same attribute (i.e., no two tuples can be projected to a single point, in any dimension), then the skyline of r is fully determined by the order in which tuples appear in each possible projection over a skyline attribute, and so is the skyline size. Without lack of generality we can substitute each attribute value with its rank, as determined by the corresponding projection. Therefore, a random relation instance r that respects the conditions (i) and (ii) can be seen as a set of d statistically independent random orders. For any r we know there is exactly one tuple having the worst rank w.r.t. the first attribute A_1. Denote by t^* this tuple. Since t^* cannot dominate any other tuple in r, $\ell_{d,n}$ is given by the expected number of skyline points in $r - \{t^*\}$ (i.e., $\ell_{d,n-1}$) plus the probability that t^* is a skyline point itself. In order to be non-dominated in r, t^* needs to be a skyline point w.r.t. attributes in $\mathcal{A} - \{A_1\}$. The expected number of such skyline points is $\ell_{d-1,n}$, and every tuple in r has the same probability of being one of those.

[7] showed that $\ell_{d,n}$ is $\mathcal{O}((\ln n)^{d-1})$; later [9] provided the closed form

$$\ell_{d,n} = \sum_{k=1}^{n} (-1)^{k+1} \cdot \binom{n}{k} \frac{1}{k^{d-1}} \tag{2}$$

and proved the tighter bound $\Theta((\ln n)^{d-1}/(d-1)!)$.

When the assumption (ii) above is dropped, i.e., the same attribute value can appear multiple times in r, the analysis becomes more complex and leads to some unexpected results [22]. A first observation is that the distribution of values over their domains

now matters, with deviations from the uniform case that lead to reducing the skyline size. A second phenomenon can be observed when values are *binned* and all values within the same bin are considered to be equal when testing dominance (which is equivalent to changing the size of the attribute domains). If $t \succ^{pto} s$ *before* binning, then reducing the domain size can lead to $t \not\succ^{pto} s$ only if the two tuples come to share the same bin on *all* the coordinates, i.e., they become equal on all skyline attributes. On the other hand, if $t \not\succ^{pto} s$ before binning, it is possible to have $t \succ^{pto} s$ (or $s \succ^{pto} t$) if t and s fall in the same bin on all the coordinates in which t (s, respectively) was worse. The larger is the number of dimensions d, the more the second effect will prevail over the first, unless the number of bins is very small. Thus, limiting the size of attributes' domains generally reduces the skyline size (since more tuples will be dominated).

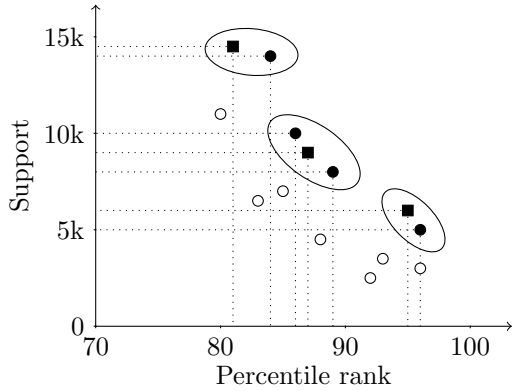

Figure 3: Diverse skyline. Most representative items are depicted as squares; ovals pair each skyline point with the closest representative item.

5.2 Representative skylines

As can be seen from the earlier discussion, skylines can be very large. In the worst case, a skyline can be as big as its originating dataset; in the average case, the size of the skyline grows with the number of its dimensions. There are several situations where dealing with a huge skyline may be impractical: the users may not be able or willing to browse through thousands of tuples; also, the time spent delivering the results may adversely affect the general user experience. Furthermore, a large skyline hardly provides any help to a user making a decision. Hence, in many real-life applications, we would like to return only a limited number κ of results that are representative of the whole skyline.

Diverse skylines. The problem of identifying the κ most representative items of the skyline has been addressed in several papers. Tao et al. [46] propose to model it as the κ-*center* problem: they suggest to select representative items by minimizing the *representation error*, i.e. the maximum Euclidean distance between a skyline point and its closest representative. The subset of points obtained this way is called a *diverse skyline*. Even if computing a diverse skyline with more than two dimensions is NP-hard, a simple greedy algorithm, like the one proposed by Gonzalez [25], can provide a 2-approximate solution in $\mathcal{O}(\kappa \cdot |sky_{\mathcal{A}}|)$. By using the Euclidean distance as a metric for measuring the similarity between skyline points, this approach implicitly assumes that all dimensions are equally important (for example: one unit of financial support is as important as one unit in the universities ranking scale). As a consequence, diverse skylines are not invariant w.r.t. scaling. Figure 3 shows the $\kappa = 3$ most representative items from the `School` relation of Example 1.1.

Top-κ RSP. An alternative approach is proposed by Lin et al. [36]: a set of κ representative items is selected in order to maximize the number of data points that are dominated by at least one of them. That is, we want to select the κ elements of the skyline that together cover the largest part of the dataset. The computation of top-κ representative skyline points (top-κ RSP) is NP-hard even in three dimensions; nevertheless, it can be easily translated into the *maximum coverage* problem and an $(1-\frac{1}{e})$-approximate solution can be obtained by applying the corresponding greedy algorithm. Top-κ RSP are invariant w.r.t. scaling and shifting operations, but they strongly depend on the points that are not part of the skyline.

Threshold-based preferences. Das Sarma et al. [43] exploit additional user preferences for selecting the most representative skyline points. They assume each user expresses her willingness to click on a particular result using threshold constraints; they also propose a probabilistic framework for modeling these preferences. In the simple case of deterministic preferences, a threshold can be represented as a point in the data space: for example, a student may be interested only in those universities that are in the 80th percentile and offer at least 20k dollars of financial support. It is easy to see that a skyline point satisfies a threshold if it dominates the corresponding point. Therefore, selecting κ skyline items that maximize the number of thresholds covered is a problem very similar to computing top-κ RSP. The same properties for invariance and complexity hold for both the approaches.

6. BEYOND PARETO DOMINANCE

6.1 Generalizations

The concept of Pareto dominance is quite restrictive. Presumably, users would like to have a richer language for formulating their preferences. Also, generalizing Pareto dominance should yield stronger dominance relations, and thus, fewer nondominated tuples and smaller query results. Below, we describe several such generalizations [41].

Grouped dominance. This variant of Pareto dominance requires that subspace dominance with respect to a set of attributes X_1 be applied only to the tuples with identical values in a set of attributes X_2, disjoint from X_1. This achieves the effect of grouping the tuples by X_2. Dominance and skylines are defined group-by-group.

k-dominance. Another variant of subspace dominance does not fix the subspace but rather considers all subspaces with cardinality k (usually $k < d$). For k-dominance of a tuple t over another tuple s it is sufficient that $t[A_{j_1} \cdots A_{j_k}]$ Pareto-dominates $s[A_{j_1} \cdots A_{j_k}]$ over *some* attributes $A_{j_1} \cdots A_{j_k}$. This concept of dominance is especially suited to applications with a very large number of dimensions, as in recommender systems (each user is a separate dimension). There, Pareto dominance (over all attributes) may occur rarely, while dominance over only some k attributes may be more common and thus more useful.

p-dominance [31, 38]. The notion of p-dominance (where "p" stands for "prioritized") builds on the algebraic definition of Pareto dominance. A different binary accumulation operator, &, is proposed. It is supposed to capture the relative importance of different attributes. p-dominance relations can now be defined using an arbitrary nesting of $\otimes$ and &. Let $\succ_X$ (resp. $\succ_Y$) be a p-dominance relation over a set of attributes X (resp. Y). Then their prioritized accumulation $\succ_{XY}^{pr} = (\succ_X \,\&\, \succ_Y)$ is defined as

$$\begin{aligned} t[XY] \succ_{XY}^{pr} s[XY] \;\equiv\; & t[X] \succ_X s[X] \\ \vee\; & t[X] = s[X] \wedge t[Y] \succ_Y s[Y] \end{aligned}$$

for $XY \subseteq \mathcal{A}$ and $X \cap Y = \emptyset$. Note that

$$\succ_{\mathcal{A}}^{pr} = >_{A_1} \,\&\, >_{A_2} \,\&\, \cdots \,\&\, >_{A_d}$$

is a *lexicographic* order over $\mathcal{U}$.

Properties. Clearly, subspace dominance, grouped dominance and p-dominance are irreflexive and transitive. However, k-dominance, being also irreflexive, is not transitive in general.

EXAMPLE 6.1. *Assuming larger values are better, the tuple $t_1 = (1, 2)$ 1-dominates the tuple $t_2 = (2, 1)$, which in turn 1-dominates $t_3 = (1, 3)$. However, t_1 does not 1-dominate t_3.*

6.2 Dominance in other spaces

Pareto dominance is often used to define dominance in a different space. We consider here dynamic and aggregate skyline queries.

Dynamic skyline queries [41]. Assume there are m functions $f_1, \ldots, f_m$, each defined over some subset of $\mathcal{A}$. We can construct a transformed tuple $f(t) = (f_1(t), \ldots f_m(t))$ for every tuple t. Now t dominates s if $f(t)$ Pareto-dominates $f(s)$. A common application involves functions capturing the 2D distance of a moving point from some fixed locations: a point x dominates another point y if for every given location z, x is not farther from z than y is.

Aggregate skyline queries. Tuples of function values (profiles) provide also a succinct way to represent aggregate properties of sets, e.g. cardinality or minimum value. The functions map sets to scalar values. The space of sets may consist of all k-element subsets of a given set of tuples [28, 35, 52], or all the tuple groups in a set of tuples [2]. Now a set T dominates another set S if the profile of T Pareto-dominates the profile of S. This approach to set dominance can be generalized to arbitrary dominance relations [52].

7. FURTHER DIRECTIONS

7.1 Skylines for uncertain data

Generalizations of the concepts of Pareto dominance and skyline to the case of uncertain databases have recently been attempted according to three different approaches, which we briefly describe here. Common to all of them is the underlying model of uncertainty, based on *probabilistic tuples* and *possible world semantics*. In short, each tuple $t \in r$ has an associated existence probability, $p(t)$, and correlations among tuples are captured by a set $\mathcal{G}$ of *generation rules*. The probabilistic relation r is seen as representing a set of standard relations, each of them termed a possible world. A possible world W is a subset of tuples from r, that respect all the generation rules in $\mathcal{G}$, and its probability $\Pr(W)$ is the probability that all and only the tuples in W indeed exist.

Both [42] and [50] view r as representing a set of uncertain objects, $O_1, \ldots, O_m$, and for each object O_i there is a mutual exclusion rule $G_i \in \mathcal{G}$ stating which are the tuples representing O_i's distribution. Given a possible world W and a tuple $t \in W$, one can determine whether $t \in sky_{\mathcal{A}}(W)$ or not. Con-

sequently, [42] defines the *skyline probability* of t as

$$\Pr_{sky}(t) = \sum_{W:t \in sky_{\mathcal{A}}(W)} \Pr(W),$$

and that of an object O as the sum of the probabilities of its tuples, $\Pr_{sky}(O) = \sum_{t_j \in O} \Pr_{sky}(t_j)$. The p-skyline of r, where p is a threshold probability, is the set of objects O such that $\Pr_{sky}(O) \geq p$. This approach has also been adopted by [6] in the context of recommender systems based on collaborative filtering.

The approach of [50] is based on the concept of *usual stochastic order*: given two random variables O_1 and O_2, O_1 stochastically dominates O_2 if for each point x in the domain of definition, the cumulative distribution of O_1 at x, $O_1.cdf(x)$, satisfies $O_1.cdf(x) \geq O_2.cdf(x)$, with strict inequality for at least one x. In the multidimensional case and discrete distributions, the cumulative distribution of object O at point x is the sum of probabilities of the instances (i.e., tuples) of O that dominate x (or coincide with it). [50] also describes an alternative approach based on the lower orthant order.

Finally, [5] introduces the concept of P-domination, i.e., domination among probabilistic tuples, and consequently defines the skyline of a probabilistic relation r as the set of those tuples that are not P-dominated. The idea builds on works dealing with the ranking of probabilistic tuples for answering top-k queries, e.g., [51]. There, one considers that the tuples are ordered by a monotone scoring function f and a specific ranking semantics is adopted to properly combine scores and probabilities. For a given ranking semantics (that is a parameter for P-domination) one stipulates that t P-dominates s if t is ranked not worse than s whatever the monotone scoring function f is.

7.2 Elicitation

A skyline query requires minimal user input: a designation of the skyline space $\mathcal{A}$ and the associated attribute domains. The orderings associated with the attributes are standard. If a user provides more information, however, it is possible to construct queries that more completely reflect her intentions.

[29] propose to use superior and inferior examples to determine missing attribute orderings in an incomplete Pareto dominance specification. The superior examples POS are the tuples that have to be in the skyline, and the inferior examples NEG, those that should not be in the skyline because of being dominated by a skyline element. The authors show that determining the existence of a strict partial order (resp. minimal strict partial order) on an attribute domain that satisfies the above POS/NEG conditions is NP-complete (resp. NP-hard). They provide greedy heuristic algorithms, without any guarantees on their performance.

[38] also use superior and inferior examples, but for a different purpose. Assuming that attribute orderings are given, their approach seeks to find a maximal p-dominance relation that satisfies the above POS/NEG conditions. They show that the associated existence problem is NP-complete but becomes polynomial if only superior examples are given.

It remains to be seen if richer forms of user input, for example dominance relationships (*does tuple t dominate tuple s?*), could be used for eliciting complete specification of dominance relations. This is a problem similar to *learning orders from examples* [14].

8. CONCLUSIONS

We believe that skyline queries provide a useful, practical, and flexible query framework for decision-making applications dealing with large amounts of data. In this paper, we have tried to showcase some of the many technical results obtained in this area. Clearly, many further research opportunities still exist or remain to be identified. For example, the connections to decision theory largely remain to be explored. Although it was not the main focus of the paper, skyline queries are a specific yet important case of preference queries, on which a large body of interdisciplinary literature exists [10, 20, 30, 31].

Among the many topics that we were unable to cover here are reverse skylines [15], skylines over joins [49], stream skylines [47], spatial skylines over moving objects [44], skylines with trade-offs [37], approximate skylines [32], and skylines in metric spaces [21]. Together with the topics discussed in this paper, they bear witness to the continuing vitality of the research on skyline queries.

9. ACKNOWLEDGMENT

The detailed comments of the anonymous reviewer are gratefully acknowledged.

10. REFERENCES

[1] F. Afrati, P. Koutris, D. Suciu, and J.D. Ullman. Parallel Skyline Queries. In *International Conference on Database Theory (ICDT)*, pages 274–284, 2012.

[2] S. Antony, P. W, D. Agrawal, and A. El Abbadi. Aggregate Skyline: Analysis for Online Users. In *Symposium on Applications and the Internet (SAINT)*, pages 50–56. IEEE, 2009.

[3] W-T. Balke, U. Güntzer, and J. X. Zhang. Efficient Distributed Skylining for Web Information Systems. In *International Conference on Extending Database Technology (EDBT)*, pages 256–273, 2004.

[4] I. Bartolini, P. Ciaccia, and M. Patella. Efficient Sort-Based Skyline Evaluation. *ACM Transactions on Database Systems*, 33(4), 2008.

[5] I. Bartolini, P. Ciaccia, and M. Patella. The Skyline of a Probabilistic Relation. *IEEE Transactions on Knowledge and Data Engineering*, 25(7):1656–1669, 2013.

[6] I. Bartolini, Z. Zhang, and D. Papadias. Collaborative Filtering with Personalized Skylines. *IEEE Transactions on Knowledge and Data Engineering*, 23(2):190–203, 2011.

[7] 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. *Journal of the ACM*, 25(4):536–543, 1978.

[8] S. Börzsönyi, D. Kossmann, and K. Stocker. The Skyline Operator. In *IEEE International Conference on Data Engineering (ICDE)*, pages 421–430, 2001.

[9] C. Buchta. On the Average Number of Maxima in a Set of Vectors. *Information Processing Letters*, 33:63–65, 1989.

[10] J. Chomicki. Preference Formulas in Relational Queries. *ACM Transactions on Database Systems*, 28(4):427–466, December 2003.

[11] J. Chomicki. Semantic Optimization Techniques for Preference Queries. *Information Systems*, 32(5):660–674, 2007.

[12] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting. In *IEEE International Conference on Data Engineering (ICDE)*, pages 717–719, 2003.

[13] J. Chomicki, P. Godfrey, J. Gryz, and D. Liang. Skyline with Presorting: Theory and Optimizations. In *Intelligent Information Systems*, pages 595–604. Springer, Advances in Soft Computing, 2005. Full version with proofs: Technical Report CS-2002-04, October 2002, Department of Computer Science, York University.

[14] W. W. Cohen, R. E. Schapire, and Y. Freund. Learning to Order Things. *Journal of Artificial Intelligence Research*, 10:243–270, 1999.

[15] R. Dellis and B. Seeger. Efficient Computation of Reverse Skyline Queries. In *International Conference on Very Large Data Bases (VLDB)*, pages 291–302, 2007.

[16] M. Ehrgott. *Multicriteria Optimization*. Springer, 2nd edition, 2005.

[17] J. Pei et al. Towards Multidimensional Subspace Skyline Analysis. *ACM Transactions on Database Systems*, 31(4):1335–1381, 2006.

[18] R. Fagin. Combining Fuzzy Information from Multiple Systems. *Journal of Computer and System Sciences*, 58(1):83–99, 1999.

[19] R. Fagin, A. Lotem, and M. Naor. Optimal Aggregation Algorithms for Middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003.

[20] P. C. Fishburn. Preference Structures and their Numerical Representations. *Theoretical Computer Science*, 217:359–383, 1999.

[21] D. Fuhry, R. Jin, and D. Zhang. Efficient Skyline Computation in Metric Space. In *International Conference on Extending Database Technology (EDBT)*, pages 1042–1051, 2009.

[22] P. Godfrey. Skyline Cardinality for Relational Processing. In *International Symposium on Foundations of Information and Knowledge Systems (FOIKS)*, pages 78–97. Springer-Verlag 2942, 2004.

[23] P. Godfrey, R. Shipley, and J. Gryz. Maximal Vector Computation in Large Data Sets. In *International Conference on Very Large Data Bases (VLDB)*, pages 229–240, 2005.

[24] P. Godfrey, R. Shipley, and J. Gryz. Algorithms and Analyses for Maximal Vector Computation. *VLDB Journal*, 16:5–28, 2007.

[25] T. F. Gonzalez. Clustering to Minimize the Maximum Intercluster Distance. *Theoretical Computer Science*, 38:293–306, 1985.

[26] B. Hafenrichter and W. Kießling. Optimization of Relational Preference Queries. In *Australasian Database Conference (ADC)*, pages 175–184, 2005.

[27] K. Hose and A. Vlachou. A Survey of Skyline Processing in Highly Distributed Environments. *The VLDB Journal*, 21(3):359–384, 2012.

[28] H. Im and S. Park. Group Skyline Computation . *Information Sciences*, 188(0):151 – 169, 2012.

[29] B. Jiang, J. Pei, X. Lin, D. W. Cheung, and J. Han. Mining Preferences from Superior and Inferior Examples. In *KDD*, pages 390–398, 2008.

[30] S. Kaci. *Working with Preferences: Less is More*. Springer, 2011.

[31] W. Kießling. Foundations of Preferences in Database Systems. In *International Conference on Very Large Data Bases (VLDB)*, pages 311–322, 2002.

[32] V. Koltun and C. H. Papadimitriou. Approximately Dominating Representatives. *Theoretical Computer Science*, 371(3):148 – 154, 2007.

[33] H. T. Kung, F. Luccio, and F. P. Preparata. On Finding the Maxima of a Set of Vectors. *Journal of the ACM*, 22(4):469–476, 1975.

[34] K.C.K. Lee, B. Zheng, H. Li, and W.-C. Lee. Approaching the Skyline in Z Order. In *International Conference on Very Large Data Bases (VLDB)*, pages 279–290, 2007.

[35] C. Li, N. Zhang, N. Hassan, S. Rajasekaran, and G. Das. On Skyline Groups. In *International Conference on Information and Knowledge Management (CIKM)*, pages 2119–2123, 2012.

[36] X. Lin, Y. Yuan, Q. Zhang, and Y. Zhang. Selecting Stars: the k Most Representative Skyline Operator. In *IEEE International Conference on Data Engineering (ICDE)*, pages 86–95, 2007.

[37] C. Lofi, U. Güntzer, and W-T. Balke. Efficient Computation of Trade-Off Skylines. In *International Conference on Extending Database Technology (EDBT)*, pages 597–608, 2010.

[38] D. Mindolin and J. Chomicki. Preference Elicitation in Prioritized Skyline Queries. *VLDB Journal*, 20(2):157–182, 2011. Special issue: selected papers from VLDB 2009.

[39] M.D. Morse, J.M. Patel, and H.V. Jagadish. Efficient Skyline Computation over Low-Cardinality Domains. In *International Conference on Very Large Data Bases (VLDB)*, pages 267–278, 2007.

[40] D. Papadias, Y. Tao, G. Fu, and B. Seeger. An Optimal and Progressive Algorithm for Skyline Queries. In *ACM SIGMOD International Conference on Management of Data*, pages 467–478, 2003.

[41] D. Papadias, Y. Tao, G. Fu, and B. Seeger. Progressive Skyline Computation in Database Systems. *ACM Transactions on Database Systems*, 30(1):41–82, 2005.

[42] J. Pei, B. Jiang, X. Lin, and Y. Yuan. Probabilistic Skylines on Uncertain Data. In *VLDB*, pages 15–26, 2007.

[43] A. Das Sarma, A. Lall, D. Nanongkai, R. J. Lipton, and J. Xu. Representative Skylines Using Threshold-Based Preference Distributions. In *IEEE International Conference on Data Engineering (ICDE)*, pages 387–398, 2011.

[44] M. Sharifzadeh and C. Shahabi. The Spatial Skyline Queries. In *International Conference on Very Large Data Bases (VLDB)*, pages 751–762, 2006.

[45] C. Sheng and Y. Tao. Finding Skylines in External Memory. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 107–116, 2011.

[46] Y. Tao, L. Ding, X. Lin, and J. Pei. Distance-Based Representative Skyline. In *IEEE International Conference on Data Engineering (ICDE)*, pages 892–903, 2009.

[47] Y. Tao and D. Papadias. Maintaining Sliding Window Skylines on Data Streams. *IEEE Transactions on Knowledge and Data Engineering*, 18(3):377–391, 2006.

[48] G. Trimponias, I. Bartolini, D. Papadias, and Y. Yang. Skyline Processing on Distributed Vertical Decompositions. *IEEE Transactions on Knowledge and Data Engineering*, 25(4):850–862, 2013.

[49] A. Vlachou, Ch. Doulkeridis, and N. Polyzotis. Skyline Query Processing over Joins. In *ACM SIGMOD International Conference on Management of Data*, pages 73–84, 2011.

[50] W. Zhang, X. Lin, Y. Zhang, M. A. Cheema, and Q. Zhang. Stochastic Skylines. *ACM Transactions on Database Systems*, 37(2), 2012.

[51] X. Zhang and J. Chomicki. Semantics and Evaluation of Top-k Queries in Probabilistic Databases. *Distributed and Parallel Databases*, 26(1):67–126, 2009.

[52] X. Zhang and J. Chomicki. Preference Queries over Sets. In *IEEE International Conference on Data Engineering (ICDE)*, pages 1019–1030, April 2011.